

## \* NOTICES \*

Japan Patent Office is not responsible for any damages caused by the use of this translation.

1. This document has been translated by computer. So the translation may not reflect the original precisely.
2. \*\*\*\* shows the word which can not be translated.
3. In the drawings, any words are not translated.

---

 CLAIMS
 

---

## [Claim(s)]

- (1) characterized by providing the following -- the equipment in the digital computer which makes the object from the 2nd object system usable by the mechanism of the 1st object system A description means to offer the related class of the 2nd object system, and description of a related type. A proxy means to be the object of the 1st object system and to generate the proxy object corresponding to the object of the 2nd object system. A forwarding means to send out operation of the proxy object of the 1st object system by using description to the object to which the 2nd object system corresponds.
- (2) Are equipment according to claim 1 and make usable the object from arbitrary numbers of other object systems by the mechanism of the 1st object system. The (a) aforementioned proxy means includes a means to generate two or more proxy objects. Each proxy object is an object of the object system of the above 1st, and corresponds to the object of one object system in other object systems. The (b) aforementioned forwarding means includes a means to send out one proxy object operation in those proxy objects to the object to which one object system in other object systems corresponds.
- (3) It is equipment according to claim 1, and the aforementioned equipment makes the object from the 1st object system usable by the mechanism of arbitrary numbers of object systems.
- (4) It is equipment according to claim 2, and the aforementioned equipment includes a means to add or remove not compile of the aforementioned equipment but the support for an object system.
- (5) It is equipment according to claim 3, and the aforementioned equipment includes a means to add or remove not compile of the aforementioned equipment but the support for an object system.
- (6) It is equipment according to claim 3, and the aforementioned equipment makes usable software which is not performed by one object system by the mechanism of arbitrary numbers of other object systems.
- It is equipment according to claim 1. (7) The aforementioned forwarding means A means to make control flow shift to the aforementioned equipment from the software which starts operation, A means to call according to the convention of the object system of the above 1st, and to search semantic information from a stack, A means to arrange semantic information to a call stack using description according to the convention of the 2nd object system A means to perform operation of a corresponding object If there is a result A means to return the result to the software which starts operation is included further.
- (8) It is equipment according to claim 7, and the aforementioned proxy means includes further the means which relates a proxy object with description of the object to which it corresponds in the 2nd object system.
- (9) A means to be equipment according to claim 8 and to arrange the aforementioned semantic information A means to traverse description which is not compiled by the aforementioned equipment A means to call semantic information according to such description, and to arrange to a stack is included further.
- (10) A means to be equipment according to claim 7 and to arrange the aforementioned semantic information A means to change into the semantic corresponding type and corresponding language type

in the 2nd object system the semantic type and language type from the semantic type used by the object system of the above 1st and a language type is included further.

(11) A means to be equipment according to claim 10 and to change the aforementioned semantic type and a language type A means to change between object types The means which carries out the trigger of the generation of a new proxy object is included further.

It is equipment according to claim 1. (12) The aforementioned equipment A means to perform mapping between basic call mechanisms, At least one side is further provided among meanses to perform mapping between low call conventions. A means to perform mapping between [ aforementioned ] low call conventions A means to perform mapping between differences of meaning type A means to constitute a proxy object dynamically if needed, A means to perform mapping between differences of error and an exception It is used combining at least one of meanses to perform mapping, between the differences for an inquiry of object information.

(13) It is equipment according to claim 1, and the aforementioned equipment possesses further a means to perform mapping between the basic call mechanisms from which an object system differs.

(14) It is equipment according to claim 1, and the aforementioned equipment possesses further a means for it to be used combining a means to perform mapping between differences of a language type, and to perform mapping between basic call mechanisms.

(15) It is equipment according to claim 1, and although the aforementioned equipment is offered in the 2nd object system, it possesses further a means to use the feature which is not offered, in the object system of the above 1st.

(16) It is equipment according to claim 1, and the aforementioned equipment is offered by the object system of the above 1st, a means to use the feature which requires the functionality of a proxy object is provided further, and such functionality is not performed depending on the object to which it corresponds in the 2nd object system.

(17) A means to be equipment according to claim 16 and to use the aforementioned feature Means related with one or more elements of description of one object The front stirrup of execution of one operation includes further a means to commission the object which had execution of the feature by the proxy associated the back or instead of execution of the operation.

(18) It is equipment according to claim 1, and a corresponding object is performed using interpretation language environment or run time.

(19) It is equipment according to claim 2. Since application or an object class is constituted Application as usual or object class composition environment is provided further. this A means to subclass-ize two or more object classes from arbitrary numbers of object systems, A means to use or incorporate the object class from arbitrary numbers of object systems At least one of the meanses which generate and incorporate the object instance from arbitrary numbers of object systems is included.

(20) It is equipment according to claim 19, and the aforementioned equipment constitutes the object which it enables to be used by the mechanism of arbitrary numbers of object systems.

(21) It is equipment according to claim 2, and provide further a means to compose two or more objects and classes from an object system to a simplification viewing space.

(22) It is equipment according to claim 9, and the aforementioned equipment enables relocation in an object class and the application process of an object, a server process, and an object system.

(23) It is the method of the digital computer which makes the object from the 2nd object system usable by the mechanism of the 1st object system. The related class of the 2nd object system and description of a related type are offered. It is the object of the 1st object system, and the proxy object corresponding to the object in the object system of the above 2nd is generated. By using the aforementioned description How to send out operation of the proxy object of the object system of the above 1st to the object to which the 2nd object system corresponds.

---

[Translation done.]

(19) 日本国特許庁 (J P)

(12) 公表特許公報 (A)

(11) 特許出願公表番号

特表平10-505693

(43) 公表日 平成10年(1998) 6月2日

(51) Int.Cl.<sup>6</sup>

G 0 6 F 9/44

識別記号

5 3 0

F I

G 0 6 F 9/44

5 3 0 M

審査請求 未請求 予備審査請求 有 (全 235 頁)

(21) 出願番号 特願平8-509776  
(86) (22) 出願日 平成7年(1995) 9月15日  
(85) 翻訳文提出日 平成9年(1997) 3月17日  
(86) 国際出願番号 PCT/CA 9 5 / 0 0 5 1 3  
(87) 国際公開番号 WO 9 6 / 0 8 7 6 5  
(87) 国際公開日 平成8年(1996) 3月21日  
(31) 優先権主張番号 0 8 / 3 0 6 , 4 8 1  
(32) 優先日 1994年9月15日  
(33) 優先権主張国 米国 (U S)

(71) 出願人 ビジュアル エッジ ソフトウェア リミ  
テッド  
カナダ国 ケベック州 エイチ4アール  
1 ブイ4 サン ローレン, コウト ヴァ  
ーチュ 3950 スイート 100  
(72) 発明者 フーディ, ダニエル, エム.  
カナダ国 ケベック州 エイチ3エイチ  
2 エヌ4 モントリオール, ディ メイソ  
ーニュ 625 アパートメント ビーエイ  
チ 211  
(74) 代理人 弁理士 大塚 康德 (外1名)

最終頁に続く

(54) 【発明の名称】 異種オブジェクトシステム相互間にインタオペラビリティを提供するシステム及び方法

(57) 【要約】

好ましい一実施例に従ったシステム及び方法は、デジタルコンピュータにおける2つ以上の異種オブジェクトシステムからのオブジェクトの相互動作を可能にし、それらを組み合わせてより大型のオブジェクト指向ソフトウェアプロジェクトを生成することができる。また、そのようなシステムと方法の用途を挙げる。外部オブジェクトシステムからのオブジェクトは変更されず、それらが使用又はアクセスされるオブジェクトシステムに対してネイティブであるように見える。実外部オブジェクトに対してネイティブプロキシオブジェクト（他のネイティブオブジェクトとの区別は不可能である）を構成する。プロキシオブジェクトは実オブジェクトに対する識別子と、そのオブジェクトをいかにしてアクセスし且つ操作すべきか、たとえば、その方法をいかにして呼び出し、その特性をいかにしてセットし、例外をいかにして処理するかのソフトウェア記述を指示するポインタとを含む。プロキシオブジェクトが操作されるとき、それはソフトウェア記述の中の命令に従うので、その結果、外部オブジェクトもそれに対応して操作される。

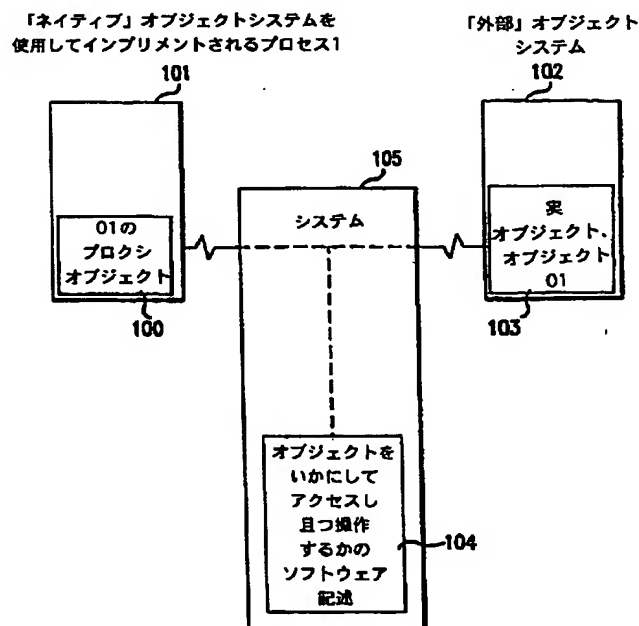


FIG.1

(2)

## 【特許請求の範囲】

(1) 第2のオブジェクトシステムからのオブジェクトを第1のオブジェクトシステムのメカニズムにより使用可能にするデジタルコンピュータにおける装置において、

第2のオブジェクトシステムの関連クラス及び関連タイプの記述を提供する記述手段と、

第1のオブジェクトシステムのオブジェクトであり且つ第2のオブジェクトシステムのオブジェクトに対応するプロキシオブジェクトを生成するプロキシ手段と、

記述を利用することにより、第1のオブジェクトシステムのプロキシオブジェクトの操作を第2のオブジェクトシステムの対応するオブジェクトへ送り出すフォワーディング手段とを具備する装置。

(2) 請求項1に記載の装置であって、任意の数の他のオブジェクトシステムからのオブジェクトを第1のオブジェクトシステムのメカニズムにより使用可能にし、

(a) 前記プロキシ手段は複数のプロキシオブジェクトを生成する手段を含み、各プロキシオブジェクトは前記第1のオブジェクトシステムのオブジェクトであり且つ他のオブジェクトシステムの中の1つのオブジェクトシステムのオブジェクトに対応しており、

(b) 前記フォワーディング手段は、それらのプロキシオブジェクトの中の1つのプロキシオブジェクト操作を他のオブジェクトシステムの中の1つのオブジェクトシステムの対応するオブジェクトへ送り出す手段を含む。

(3) 請求項1に記載の装置であって、前記装置は第1のオブジェクトシステムからのオブジェクトを任意の数のオブジェクトシステムのメカニズムにより使用可能にする。

(4) 請求項2に記載の装置であって、前記装置は、前記装置のコンパイルでなくオブジェクトシステムに対する支援を追加又は除去する手段を含む。

(5) 請求項3に記載の装置であって、前記装置は、前記装置のコンパイルでな

(3)

くオブジェクトシステムに対する支援を追加又は除去する手段を含む。

(6) 請求項3に記載の装置であって、前記装置は、1つのオブジェクトシステムで実行されないソフトウェアを任意の数の他のオブジェクトシステムのメカニズムにより使用可能にする。

(7) 請求項1に記載の装置であって、前記フォワーディング手段は、  
制御の流れを、操作を開始するソフトウェアから前記装置へ移行させる手段と

、  
前記第1のオブジェクトシステムのコンベンションに従って呼び出しスタックから意味情報を検索する手段と、

第2のオブジェクトシステムのコンベンションに従い且つ記述を利用して、呼び出しスタックに意味情報を配置する手段と、

対応するオブジェクトの操作を実行する手段と、

結果があれば、その結果を、操作を開始するソフトウェアに戻す手段とを更に含む。

(8) 請求項7に記載の装置であって、前記プロキシ手段は、プロキシオブジェクトを第2のオブジェクトシステムの中の対応するオブジェクトの記述と関連づける手段を更に含む。

(9) 請求項8に記載の装置であって、前記意味情報を配置する手段は、

前記装置にコンパイルされない記述をトラバースする手段と、

そのような記述に従って意味情報を呼び出しスタックに配置する手段とを更に含む。

(10) 請求項7に記載の装置であって、前記意味情報を配置する手段は、

前記第1のオブジェクトシステムで使用する意味タイプ及び言語タイプからの意味タイプ及び言語タイプを第2のオブジェクトシステム内の対応する意味タイプ及び言語タイプに変換する手段を更に含む。

(11) 請求項10に記載の装置であって、前記意味タイプ及び言語タイプを変換する手段は、

オブジェクトタイプ相互間で変換する手段と、

(4)

新たなプロキシオブジェクトの生成をトリガする手段とを更に含む。

(12) 請求項1に記載の装置であって、前記装置は、

基本呼び出しメカニズム相互間でマッピングを実行する手段と、

低レベル呼び出しコンベンション相互間でマッピングを実行する手段のうち少なくとも一方を更に具備し、前記低レベル呼び出しコンベンション相互間でマッピングを実行する手段は、

意味タイプの相違の相互間でマッピングを実行する手段と、

必要に応じてプロキシオブジェクトを動的に構成する手段と、

誤り及び例外の相違の相互間でマッピングを実行する手段と、

オブジェクト情報の問い合わせに際しての相違の相互間でマッピングを実行する手段のうち少なくとも1つと組み合わせて使用される。

(13) 請求項1に記載の装置であって、前記装置は、オブジェクトシステムの異なる基本呼び出しメカニズムの間でマッピングを実行する手段を更に具備する。

(14) 請求項1に記載の装置であって、前記装置は、言語タイプの相違の相互間でマッピングを実行する手段と組み合わせて使用され且つ基本呼び出しメカニズムの相互間でマッピングを実行する手段を更に具備する。

(15) 請求項1に記載の装置であって、前記装置は、第2のオブジェクトシステムでは提供されるが、前記第1のオブジェクトシステムでは提供されない特徴を使用する手段を更に具備する。

(16) 請求項1に記載の装置であって、前記装置は、前記第1のオブジェクトシステムにより提供され、プロキシオブジェクトの機能性を要求する特徴を使用する手段を更に具備し、そのような機能性は第2のオブジェクトシステム中の対応するオブジェクトによっては実行されない。

(17) 請求項16に記載の装置であって、前記特徴を使用する手段は、

1つのオブジェクトを記述の1つ又は複数の要素と関連づける手段と、

1つの操作の実行の前又は後に、あるいはその操作の実行の代わりに、プロキシによる特徴の実行を関連づけられたオブジェクトに委託する手段とを更に含む。

(5)

。

(18) 請求項1に記載の装置であって、対応するオブジェクトは解釈言語環境又はランタイムを使用して実行される。

(19) 請求項2に記載の装置であって、

アプリケーション又はオブジェクトクラスを構成するために、従来通りのアプリケーション又はオブジェクトクラス構成環境を更に具備し、これは、

任意の数のオブジェクトシステムからの複数のオブジェクトクラスをサブクラス化する手段と、

任意の数のオブジェクトシステムからのオブジェクトクラスを利用する又は取り込む手段と、

任意の数のオブジェクトシステムからのオブジェクトインスタンスを生成し且つ組み込む手段のうち少なくとも1つを含む。

(20) 請求項19に記載の装置であって、前記装置は、任意の数のオブジェクトシステムのメカニズムにより使用されるべくイネーブルされるオブジェクトを構成する。

(21) 請求項2に記載の装置であって、複数のオブジェクトシステムからのオブジェクト及びクラスを単一化ビューイングスペースへ編成する手段を更に具備する。

(22) 請求項9に記載の装置であって、前記装置は、オブジェクトクラス及びオブジェクトのアプリケーションプロセス、サーバプロセス及びオブジェクトシステムの中での再配置を可能にする。

(23) 第2のオブジェクトシステムからのオブジェクトを第1のオブジェクトシステムのメカニズムにより使用可能にするデジタルコンピュータの方法であって、

第2のオブジェクトシステムの関連クラス及び関連タイプの記述を提供し、

第1のオブジェクトシステムのオブジェクトであり、且つ前記第2のオブジェクトシステム中のオブジェクトに対応するプロキシオブジェクトを生成し、

前記記述を利用することにより、前記第1のオブジェクトシステムのプロキシ

(6)

オブジェクトの操作を第2のオブジェクトシステムの対応するオブジェクトへ送り出す方法。

(7)

## 【発明の詳細な説明】

異種オブジェクトシステム相互間にインタオペラビリティを提供するシステム及び方法

## 技術分野

本発明はデジタルコンピュータのオブジェクト指向（オブジェクト・オリエンティッド）ソフトウェアシステム及びそれに関連する方法に関する。

## 背景技術

オブジェクト指向ソフトウェア技法を使用すると、ソフトウェアオブジェクトを組み合わせることにより、デジタルコンピュータのソフトウェアアプリケーションが生成される。このプロセスを容易にするため、オブジェクト指向ソフトウェアシステムは、通常、オブジェクトモデルと呼ばれるアーキテクチャ仕様を提供しており、これは、その仕様に合わせて開発される全てのオブジェクトを1つのアプリケーションの中で一体に境界なく作用させることができる。オブジェクトモデルの例としては、Object Management Groupのコモン・オブジェクト・リクエスト・ブローカ・アーキテクチャ（Common Object Request Broker Architecture: CORBA）やマイクロソフト（Microsoft）のコモン・オブジェクト・モデル（common Object Model: COM.）が挙げられるであろう。また、このようなシステムは、通常、オブジェクトモデル内で提供される基本特徴を実現するオブジェクトシステムと呼ばれるソフトウェアも提供する。

オブジェクトシステムは数多くあり、マイクロソフトのオブジェクト・リンキング・アンド・エンベディング（Object Linking and Embedding: OLE）（COMオブジェクトモデルに準拠する）、又はIBMのディストリビューテッド・システム・オブジェクト・モデル（Distributed System Object Model: DSOM）及びイオナ（Iona）のORBIX（共にCORBAオブジェクトモデルに準拠する）などの非常に一般的な性質を持つものもある。例えば、OLE 2 Programmers Reference, 第1巻及び第2巻、Microsoft Press社刊、1994年；IBM SOMobjects Developer Toolkit V2.0, Programmers Reference Manual, 1993年；Iona ORBIX, Advanced Programmers Guide, 1994年；及びThe Common Object request Broker: Architecture and Specification, 第6章、OMG, 1991年などを参

照。これらの参考文献は参考として本明細書にも取り入れられている。

その他のオブジェクトシステムは、例えば、ロータスノート (Lotus Notes) などのグループウェア又は関連データベースなどの領域で特定の機能性を提供するように設計されている。更に別のオブジェクトシステムは、例えば、ノーベル (Novell's) の AppWare Bus, ヒューレットパッカード (Hewlett Packard's) の Broadcast Message Server 及び Microsoft Visual Basic の V B X オブジェクトメカニズムなどのアプリケーションに特定されている。例えば、Lotus Notes Programmers Reference Manual, 1993 年; Novell Visual AppBuilder Programmers Reference Manual, 1994 年; Hewlett Packard Softbench BMS の Programmer Reference Manual, 1992 年; Microsoft Visual Basic 3.0 Professional Features Book 1, Control Development Guide, 1993 年などを参照。これらの参考文献も参考として本明細書に取り入れられている。

ソフトウェアアプリケーションを生成する場合、種々のオブジェクトシステムは、様々に異なるタスクに最も良く適合しており、また、最良の解決方法は、通常、最良の部分（すなわち、オブジェクト）から作成されるので、様々なオブジェクトシステムからオブジェクトを組み合わせることが望ましい。ところが様々なオブジェクトシステムからのオブジェクトは、当然のことながら、いくつかの理由により一体には動作しない。

使用される基本メカニズムの相違、並びにタイプ及びクラスの物理的レイアウトなどの低レベルのコール (call) のコンベンション (convention) の相違を含めて、オブジェクトを生成し、方法をコールし且つ各オブジェクトシステムで特性を設定するための手段に相違があるため、オブジェクトシステムは互換性を失う。例えば、基本レベルにおいて、COM のようないくつかのオブジェクトシステムは直接 C++ 呼び出しメカニズムを使用する。DSOM プリプロセスソースコードなどの他のシステムでは、直接呼び出しを実行するのではなく、オブジェクトシステムから機能を読み出し、その結果、実方法にポインタを返す。このポインタをデリファレンスして、実際に方法を読み出す。OLE Automation などの更に別のオブジェクトシステムは、専門化された機能を提供し、開発時にはそれらを使用して方法を読み出さなければならない（多くの場合、これをダイナミック・イ

(9)

ンボケーション・インタフェース (Dynamic Invocation Interface: D I I) と呼ぶ)。これらの機能は、呼び出すべき方法を引数として取り出すと共に、方法の引数を取り出し（通常は特定のフォーマットにパックされている）、開発者に対して方法呼び出す。基本呼び出しメカニズムには、他の広い範囲の相違や変形が数多く存在している。それらの基本メカニズムは、それぞれ、詳細な点でも異なっている。例えば、CORBAは環境ポインタ引数を必要とするが（更に、オプションのコンテキスト引数 (context argument) を有する）、他のオブジェクトシステムは必要としない。

基本呼び出しメカニズムの大きな相違に加えて、手続き呼び出しコンベンションと呼ばれることもある低レベル呼び出しコンベンションにも多くの相違点がある。例えば、戻り値のタイプがフロート又は構造である場合、異なるオブジェクトシステムは方法からの戻り値を異なる方式で処理する。ある場合には値をプロセッサスタックに戻しても良いが、別の場合には値をレジスタに導入しても良い。すなわち、異なるオブジェクトシステムからの方法の戻り値を使用すれば、その結果、誤りが生じてしまうであろう。手続き呼び出しコンベンションの相違の例としては、その他に、構造をメモリにパックする方式や、引数をスタックに配置する方式が考えられる。

様々なオブジェクトシステムは、他のオブジェクトシステムとの間では互換性を持ち得ない様々なタイプをも支援する。タイプの単純な例は、整数、フロートなどの言語タイプを含む。更に複雑な言語タイプとしては、アレイ、ストリング及びオブジェクトがある。また、CORBAのAny、COMCのVARIANTのような「可変タイプ」などの意味タイプ (semantic types) もある。意味タイプは、システムに対して特定の意味論的意味を有するという点で言語タイプとは異なる。いくつかの意味タイプは概念の上では様々なオブジェクトシステムの中で同一のものを意味するであろうが、それに対応する言語表現とインプリメンテーションが全く異なる場合もある。共通する例はストリングである。COMでは、ストリングは「BSTR」（長さ情報を含むNULLで終わらないストリング）を使用して表現されるが、CORBAの場合には、ストリングは従来通りのC言語バイトアレイ（長さ情報を伴わないNULLで終わるストリング）である。従って、COMオブ

ジ

ェクトについては、CORBAオブジェクトで使用されるストリングに対して作用するコピーや比較などの機能がいずれも実行不可能になるため、COMオブジェクトは、CORBAオブジェクトにBSTRを渡すことができないであろう。同様に、CORBAのAnyやCOMのVARIANTなどの「可変タイプ」は同一のものを「意味」してはいるが、それらは互換性を持たない。

加えて、オブジェクトシステムはライフサイクル管理に関して、互換性を持ち得ない様々な規則を有する。ライフサイクル管理という用語は、オブジェクトを生成し、記憶し、削除するときに必要とされるプロセスを表わす。例えば、COMでは、開発者はオブジェクトを自動的に削除できるようにリファレンス・カウンティングを実行しなければならない。関連データベースのライフサイクル管理ははるかに精巧であるが、CORBAのライフサイクル管理はリファレンスカウンティングを伴わず、ごく単純である。

多くの場合、オブジェクトは引数として方法に渡されるので、上記のライフサイクル管理の問題は重大である。1つのオブジェクトシステムにあるオブジェクトが外部オブジェクトシステムのオブジェクトの方法を呼び出し、（そのオブジェクトシステムからの）オブジェクトを引数として方法に渡すという場合を考えてみる。外部オブジェクトシステムはそれ自身のオブジェクトしか理解しないので、オブジェクト引数を外部オブジェクトシステムの対応するオブジェクトに動的に変換しなければならない。言い換えれば、引数として渡される元のオブジェクトに適合させるべく、外部オブジェクトシステムで新たなオブジェクトを生成しなければならない。オブジェクトシステム相互間でインタオペラビリティを作用させるべきであれば、このような動的ライフサイクル管理、すなわち、対応するオブジェクト破壊を伴うオブジェクト生成を全て適正に処理しなければならない。

オブジェクトシステムのインタオペラビリティ(interoperability)のもう1つの面は、オブジェクトシステム間の例外と誤りの処理の相違である。通常、オブジェクトのコードの中で現われる誤り又は例外もそのコードを呼び出したオブジ

## (11)

ェクトの中で処理しなければならない。2つのオブジェクトが異なるオブジェクトシステムから出たものであり、誤り処理メカニズムが互換性を持たない場合、

ソフトウェア障害が起こるであろう。

オブジェクトシステムが様々に異なれば、オブジェクトに関する情報を動的に問い合わせる方法も異なる。この機能性は、汎用マクロスクリプト記録機能を提供するオブジェクトシステム並びに分散計算能力を提供するオブジェクトシステムについて要求される。例えば、マイクロソフトのOLE 2 Programmers Referenceの第2巻、アップル社のInside Macintosh: Interapplication Communicationの第8章（参考として本明細書にも取り入れられている）、又はObject Management GroupのThe Common Object Request Broker: Architecture and Specificationの第6章を参照。すなわち、メカニズムがオブジェクトに関する情報を問い合わせる方式に互換性がなければ、その結果、他のオブジェクトシステムにおいてオブジェクトを使用する方法に重大な制限が生じる。

先に述べた通り、オブジェクトシステムの設計目標はそれぞれ異なっている。従って、各オブジェクトシステムは、通常、別のオブジェクトシステムでは利用できない機能性を有する。その例を2つ挙げると、CORBAネームスペース（COMにはこれはない）と、複数のインタフェースを支援するCOMオブジェクト（CORBAはそのようなオブジェクトを持たない）である。あるオブジェクトに別のオブジェクトシステムでのみ利用可能な何らかのアクションを実行することを求めると、ソフトウェアの誤りが起こる可能性がある。

要するに、強く求められる目標は様々なオブジェクトシステムの間にインタオペラビリティを提供することである。異なるオブジェクトシステムはオブジェクトに異なる必要条件を課するので、インタオペラビリティを可能にするためには専門化されたソフトウェアシステムが必要である。従来技術にも、オブジェクトシステム相互間にインタオペラビリティを提供する方式がある。そのうちスタブ・ファンクション・ラッパ(stub function wrapper)；コモン・ワイヤ・プロトコル；ダイナミック・コンバータの3つが興味ある方式である。

スタブ・ファンクション・ラッパ方式の場合、自動化ツールを使用して、別の

オブジェクトモデルの仕様に従うコードにオブジェクトを「ラップ（包む）」するスタブ機能を生成する。すなわち、このツールはネイティブオブジェクトへ要求を送り出すために必要なコードのみを含むオブジェクトクラスを外部オブジェ

クトシステムで生成する。この「スタブ」コードをコンパイルし、アプリケーションにリンクする。スタブファンクション方式を示す良い例が、IBMのウインドウズ(Windows)用DSOMで実行されているCOMインタオペラビリティである。これを使用すると、DSOMオブジェクトを持つ開発者がそれらのオブジェクトをCOMの中から使用したいときに、自動ツールを利用して、対応するCOMオブジェクトのソースコードを生成できる。COMオブジェクトのソースコードは単にDSOMへ呼び出しを送り出す一組のスタブ機能である。例えば、COMアプリケーションがCOMオブジェクトの方法を呼び出したときには、方法はその方法のDSOMバージョンを再呼び出しするだけである。

一組のスタブ・ファンクション・ラッパを生成するための自動化ツールの1つは、2つのオブジェクトシステムの間の一方向インタオペラビリティを提供する。動的ライフサイクル管理で求められるような二方向インタオペラビリティを提供するためには、第2のツールを書き込まなければならない。追加のオブジェクトシステムを支援すべきであり、且つ3つのシステム全ての間に完全なインタオペラビリティが要求される場合には、追加の4つの自動化ツールを書き込まなければならない。実際に、N個のオブジェクトシステムの間でインタオペラビリティを成立させるために要求されるそのようなツールの数は $(N^2 - N)$ である。更に、オブジェクトクラスごとに、スタブラッパ機能の $(N^2 - N)$ 個のバージョンを生成し、管理し、支援しなければならない。ある特定のアプリケーションが数百のクラスを有していると仮定すれば、これは重大な欠陥である。また、この方法は、タイプ変換の処理、低レベル呼び出しコンベンションの相違、ライフサイクル管理の相違、誤り及び例外処理の相違、問い合わせ方式、又は機能性の相違などのインタオペラビリティに関わるその他の問題点については何の支援も行わない。そのため、オブジェクトのユーザは、手動操作によりそのような変換をユーザ自身のソフトウェアにコード化して、オブジェクトが別のオブ

ジェクトシステムのものである（すなわち、オブジェクトがネイティブオブジェクトとは明確に区別できない）ことを明らかにしなければならない。最後に、新たなオブジェクトシステムに対する支援は動的ではないので、コードを再コンパイルし、再びリンクしなければならない。

ダイナミックコンバータは、共にダイナミック・インボケーション・インタフェース（D I I）を支援する2つのオブジェクトシステムの間にインタオペラビリティを提供するように設計されている。D I Iは方法を「動的に呼び出す」。すなわち、D D Iは、開発者に対してオブジェクトの方法を動的に呼び出し、その呼び出しをアプリケーションにコンパイルすることを要求する代わりに、事前に定義済コンベンションを使用して引数を方法に渡す一組の機能である。ダイナミックコンバータは2つのD I Iの間のハードコード化マップである。方法が呼び出されると、コンバータコードは第2のオブジェクトシステムのD I Iに適するフォーマットに情報をパッケージし、次に、方法呼び出す。

スタブ・ラッパ・メカニズムの場合と同様に、この方式はN個のオブジェクトシステムを支援するために（Nの2乗－N）個のコンバータを必要とする。更に、ダイナミックコンバータはD I Iを支援するオブジェクトシステムと併用しなければ有効ではない。また、D I Iを使用するために性能が低下する。スタブラッパ方式と同様に、この方法も、タイプ変換の処理、誤り及び例外処理の相違、問い合わせ方式、又は機能性の相違などのインタオペラビリティのその他の問題について一般に支援しない。従って、ネイティブオブジェクトと区別不可能である外部オブジェクトを提供することができない。

コモン・ワイヤ・プロトコル方式は分散オブジェクトシステム、すなわち、異なる機械アーキテクチャを有する異なるコンピュータに記憶させ得るオブジェクトを含むオブジェクトシステムと併用されるように設計されている。コモン・ワイヤ・プロトコル方式の場合、基礎をなす共通の分散計算システム（D C S）、すなわちコモンワイヤを共用するオブジェクトシステムがインタオペレート（interoperate）できる。共通分散計算システムでは、D C Sにより非（Nの2乗－N）言語データタイプ転送メカニズムが形成されるので、オブジェクトシステムは

言語データタイプを転送することができる。

この方式は低レベル呼び出しコンベンションの問題に対処し、低レベルデータタイプのマッピングを実行する。N個のオブジェクトシステムを支援することがどれほど複雑になるかは、オブジェクトシステムが同一のオブジェクトモデルを共用するか否かによって直接に決まる。共用しているならば、基本呼び出しコン

ベンションの相違、意味タイプの相違、ライフサイクル管理の相違、誤り及び例外処理の相違、又は機能性の相違の間にインタオペラビリティを成立させる必要はなく、各オブジェクトシステムはDCSを支援するだけで良い。これを示す例は、それぞれがOpen Software FoundationのDistributed Computing Environment (DCE) を使用できるIBMのDSOM、イオナ(Iona)のOrbix、ヒューレットパッカートのDOMFなどの様々なCORBAオブジェクトシステム相互間のインタオペラビリティであろう。

共通オブジェクトモデルを共用しないオブジェクトシステムの間インタオペラビリティである場合には、この方式は残る問題点を処理するために(Nの2乗-N)個のコンバータを必要とする。更に、インタオペレートしなければならないオブジェクトシステムが共用分散計算システムの最上部に形成されていないときには、この方式は全く機能しない。また、DCSに依存しているため、資源の消耗がことのほか激しい。単純なデータタイプの転送は処理するが、より複雑なタイプ変換、ライフサイクル管理の相違、誤り及び例外処理の相違、問い合わせ方式、又は機能性の相違を処理するための汎用メカニズムは提供されないで、他の従来技術と同様の欠点が見られる。

オブジェクトシステムのインタオペラビリティの問題を直接解決しようとした従来の上記の方式と並んで、この問題の様々な要素をそれぞれ別の見方で処理した方法が他にもある。

異なる低レベル呼び出しコンベンションの間のマッピングの問題に対処したシステムはいくつかある。米国特許第4,736,321号には、対話言語ワークスペース、すなわち、APLが外部言語手続きを呼び出すことができる方法が記載されていた。この場合、APL環境に対してFORTRAN機能を宣言し、A

(15)

PL環境はAPL呼び出し及び引数をFORTRAN呼び出し環境にマッピングした。この特許の方法は対話APL言語環境に特定されており、APLから複数の言語（FORTRAN及びアセンブラ）への一方向アクセスを実行する。任意の数の言語の間で両方向アクセスを行う方法ではなく、コンパイル言語コードシステムの間では機能しない。

これに類似するメカニズムは米国特許第5, 274, 821号にProlog

に関して記載されており、この場合、Prolog言語手続きは外部言語機能を読み出すことができ、逆に、外部言語機能がProlog言語を読み出すこともできた。この特許においては、マッピングはテーブル駆動方式を使用して実行された。すなわち、Prologから複数の言語へのマッピング及び複数の言語からPrologへのマッピングは、テーブルの中で低レベル呼び出しコンベンションをシステムに記述することによって実行された。この情報は実行時にマッピングを動的に実行するために使用された。米国特許第4, 736, 321号の方法と同様に、この方法是对話言語環境に特定され、この場合はPrologに特定されている。PrologからN個の言語への両方向マッピングを実行するが、N個の言語相互間でのマッピングは実行しない。この方法はコンパイル言語コードシステムの間では機能せず、いずれにしても、 $(N^2 - N)$ 個の変換テーブルを必要とするであろう。

米国特許第5, 210, 876号に記載されたメカニズムにおいては、インタプリタはコンパイル手続きを読み出すことが可能であり、コンパイル手続きは解釈済手続きを読み出す。すなわち、元のコンパイル手続きにより呼び出される新たな中間コンパイル手続きを生成した。その後、新たなコンパイル手続きを元のコンパイル手続きと動的にリンクさせた。新たに生成された手続きは引数をインタプリタが要求するフォーマットに変換し、次に、解釈済手続きを読み出した。最後に、結果をコンパイル言語により要求される元の形に変換した。従来の他の技法と同様に、この特許は解釈される環境がコンパイル言語コードを読み出すのを容易にしている。また、従来の他の技法と同様に、N個のシステムの間の手続き呼び出しコンベンションの変換を支援するためには、 $(N^2 - N)$ 個のコー

ド発生器が必要であろう。

米国特許第5, 097, 533号は、様々な事前確定済コンピュータ言語を単一のソフトウェアシステムにインタフェースする方法を説明している。この場合には、言語ごとにコードを書き込み、その言語のAPIから基礎となるソフトウェアシステムの単一のAPIへマッピングする。この特許では、言語ごとにマッピングを実行するために、コードを書き込むことによりN個の言語から1つの言語へ手続き呼び出しコンベンションをマッピングする。この方法はN個の言語コ

ンベンションの間での変換には有効でなく、いずれにしても、 $(N^2 - N)$ 個のコードブロックを書き込む必要があるだろう。

米国特許第5, 146, 593号には、単一のソフトウェアインタフェースを使用して複数の手続きを呼び出す方法が記載されている。実際、これはDIIの一例にすぎず、異なるプログラミング言語へマッピングするように設計されたDDIである。この場合、方法はテーブルを使用して、特定の言語の低レベル呼び出しコンベンションを記述する。ユーザ（すなわち、呼び出しを実行したいソフトウェア開発者）はDIIインタフェースをそのアプリケーションとリンクし、手続き識別子と、事前定義済フォーマットをとるデータ構造とをアプリケーションに渡すことにより、DDIインタフェースを使用して全ての呼び出しを実行する。この方法はN個の言語呼び出しコンベンション相互間のインタオペラビリティを支援するものではなく、その目的のために拡張しようとしたならば、やはり $(N^2 - N)$ 個のテーブルが必要になるであろう。

上記の特許には、様々な言語の低レベル呼び出しコンベンションの相違を処理するためのメカニズムを提供するように設計されているということが共通していた。米国特許第5, 210, 876号を除き、それぞれの特許は様々な言語タイプを変換する手段も提供する。しかしながら、これらの特許は高レベル呼び出しコンベンションの相違という問題には触れていない。事実、それぞれの高レベル呼び出しコンベンションは異なっているため、その相違を問題にする必要はない。また、意味タイプ相互間のマッピングの支援も行っていない。

例えば、米国特許第5, 187, 787号のような他の特許は、意味タイプ互

間のマッピングの問題を扱っている。この特許においては、意味タイプ間のマッピングは、2つのソフトウェアアプリケーションをデカップルする通信インタフェースを構成するために使用される、より大型のシステムの1つの要素であるにすぎなかった。この場合、通信で使用される意味タイプは自己記述型であるように設計されていた。この特許は、インタオペラビリティを得るために意味記述をデータ表現から分離させなければならないと教示している。更に、開発者は、いずれかの場所に、別の意味形態をとって存在すると思われるデータをアクセスするためにシステムAPIを使用する。従って、このメカニズムは開発者の側にデ

カップルデータを使用するための明確な知識を必要とし、単一DIIに類似しているので、同じような制約が見られる。

米国特許第5, 278, 978号には、2つのデータベースの間で情報を転送する方法が記載されていた。システムの一部として、言語タイプと意味タイプの双方をマッピングするためのメカニズムが説明されていた。各言語タイプにはマーカ記述子と呼ばれる標準識別子が付されている。同様に、各意味タイプには、それとは別に、タイプ定義と呼ばれる標準識別子が付されている。別のデータベースからデータを受信したとき、それらの記述を使用して変換を実行する。変換は受信側データベースで実行されるので、それぞれのデータベースは他の全ての機械環境と意味タイプの事前定義済記述、すなわち、(Nの2乗-N)個の変換記述を有していなければならない。

#### 発明の概要

本発明は、いくつかの実施例の中で、複数の異種オブジェクトシステムからのソフトウェアオブジェクトが両方向にインタオペレイトし、且つより大型のソフトウェアシステムの生成に際してそれらのソフトウェアオブジェクトを組み合わせることを可能にするデジタルコンピュータ内の単一システムを提供する。外部オブジェクトシステムからのオブジェクトは変更されないが、それらが使用又はアクセスされるオブジェクトシステムに対してネイティブであるように見える。追加オブジェクトシステムへの支援はシステムの実行中に動的に追加されるので、支援される他の全てのオブジェクトシステムとの間に両方向インタオペラビリ

ティが追加される。また、オブジェクトを変更する必要なく、1つのオブジェクトシステムの特徴を他のオブジェクトシステムでも支援できるようにするメカニズムも提供される。

外部オブジェクトを別のオブジェクトシステムでも使用できるようにするために、これらの実施例のシステムは、実外部オブジェクトに対してネイティブ・プロキシ・オブジェクト (native proxy object) (他のネイティブオブジェクトとは区別不可能である) を構成する。プロキシオブジェクト (proxy object) は実オブジェクトに対する識別子と、そのオブジェクトをいかにしてアクセスし、操作するか、例えば、その方法をいかにして呼び出し、その特性をいかにしてセッ

トし、例外をいかにして処理するかを表わすソフトウェア記述に対するポイントを含む。プロキシオブジェクトを操作するときには、ソフトウェア記述の中の命令に従うので、その結果、外部オブジェクトもそれに相応して操作される。

これらの実施例のシステムは、デフォルト・インプリメンテーション (default implementations) を伴ういくつかのオブジェクト指向フレームワーク (object-oriented frameworks) を提供する。オブジェクトシステム特有の又は均一のネームスペースにおいてオブジェクトの場所を動的に指定するフレームワークと；オブジェクトクラス又はインスタンス (instance) の特性を記述するフレームワークと；意味タイプと言語タイプの双方を記述するフレームワークと；プロキシオブジェクトの操作を実外部プロジェクトへ送り出すためにソフトウェア記述を「実行」するフレームワークと；誤り及び例外を処理するフレームワークと；オブジェクトのライフサイクルを生成し、コピーし、破壊し、管理するフレームワークと；オブジェクト定義を「エクスポート」する、すなわち、1つのオブジェクトシステムのオブジェクトを一時的に又は永久に他のオブジェクトシステムにあるネイティブオブジェクトとは区別不可能なオブジェクトクラスとして見えるようにするフレームワークとがある。

更に、どのオブジェクトクラスの記述にも余分な情報を追加するフレームワークがある。そのような追加情報を「ミキシン (mixin)」と呼ぶ。オブジェクトをそれが支援しない方式で操作される場合、例えば、ある集合体の中央の要素を

戻すことが求められているが、その要素がそのような方法を支援しない場合には、システムはオブジェクトと関連するミキシンに要求を支援できるか否かを尋ねる。支援できるのであれば、ミキシンはその要求を引き継ぎ、実行する。このようにして、オブジェクトシステムのオブジェクトの能力を、それらのオブジェクトが当然支援しないと考えられる他のオブジェクトシステムの特徴をもって拡張するための汎用メカニズムが提供される。

上記のそれぞれのフレームワークのデフォルト・インプリメンテーションへの拡張はオブジェクトシステムによってグループ化され、オブジェクトシステムアダプタ (OSA) と呼ばれるライブラリにパッケージングされる。OSAをシステムのOSAレジストリ・フレームワーク (OSA Registry framework) にロードす

ることができ、従って、新たなオブジェクトシステムを動的に支援し、且つ他のオブジェクトシステムとの間に完全なインタオペラビリティを提供するために必要なあらゆるものを追加できる。

このシステムが独立して使用されても良く、あるいは、より大型のソフトウェアシステムの1つの要素として組み込まれても良いことは当業者には自明であろう。

本発明の更に別の実施例も提供される：

(A) 「汎用オブジェクト」、すなわち、複数のオブジェクトシステムで同時にインプリメントされるように見え、且つそれが支援するオブジェクトシステムを動的に変更することができる単一のオブジェクトの生成を可能にするシステム及び方法。前記システムは個別構成、アプリケーション構成及びサーバ構成でもオブジェクトの生成を可能にする。更に、そのようなオブジェクトは解釈済又はコンパイル済言語技法に基づいていても良い；

(B) ライブラリ又は対話方式に基づき、前記「汎用オブジェクト」を含むオブジェクトクラスの構成を可能にするシステム及び方法。前記オブジェクトクラスを生成するとき、このようなシステムは複数のオブジェクトシステムからのオブジェクトのサブクラスを構成するか、それらのオブジェクトを取り込むか、又はそのインスタンスを組み込んでも良い。

(20)

(C) アプリケーション及びサーバの実行中に、オブジェクトクラス及びオブジェクトを利用するソフトウェアも実行中であっても、そのソフトウェアを崩壊させることなく、1つ又は複数のオブジェクトシステムの中でオブジェクトクラス及びオブジェクトの場所をアプリケーション間及びサーバ間で変更することを可能にするシステム及び方法。

#### 図面の簡単な説明

本発明の上記の面は、添付の図面と関連して与えられる以下の詳細な説明を参照することにより更に容易に理解されるであろう。

図1は、本発明の好ましい一実施例に従った（使用中の）システムの概略図である。

図2は、本発明の好ましい一実施例に従ったシステムアーキテクチャの概略図である。

図3は、アダプタ・レジストリ・フレームワーク (Adapter registry framework) にプラグ接続された図2のオブジェクト・システム・アダプタ (Object System Adapter) を示す図である。

図4は、図2の実施例に従った1つの方法と、1つの特性とを有するクラスの記述を示す図である。

図5は、図2の実施例に従った入れ子型記述を示す図である。

図6は、CORBA C言語呼出しコンベンションに従う方法の記述を示す図である。

図7は、ダイナミック・インボケーション・インタフェース (Dynamic Invocation Interface) を使用して呼び出される類似する方法の記述を示す図である。

図8は、オブジェクトの代表的なライフサイクルを示す図である。

図9は、図2のシステムを使用してクラスを露出させるために使用されるプロセスを示す図である。

図10は、図2の実施例に従った代表的なプロキシオブジェクトを示す図である。

図11は、本発明の好ましい一実施例に従った「汎用オブジェクト」の個別構

(21)

成、アプリケーション構成及びサーバ構成を示す図である。

図12aは、個別構成で汎用オブジェクトをイネーブルするシステムの一例を示す図である。

図12b及び図12cは、それぞれ、アプリケーション構成とサーバ構成を示す図である。

#### 特定の実施例の詳細な説明

図1は、本発明の好ましい一実施例に従ったデジタルコンピュータで使用されるシステム105の簡略化した概略図を示す図である。

プロセス101は、プロセス102及びオブジェクト103が実行される第2のオブジェクトシステムとは明確に異なる、「ネイティブ」のオブジェクトシステムと呼ばれる第1のオブジェクトシステムを使用して実行される。プロセス102とオブジェクト103を実行する第2のオブジェクトシステムは、「外部(foreign)」オブジェクトシステムと呼ばれる(オブジェクト103を使用しているプロセス101に対して外部のものである)。システム105はプロクシオブジェクト100を構成している。プロクシオブジェクト101は、プロセス101中のネイティブオブジェクトシステムを使用して実行される他のオブジェクトからは区別不可能であるように見える。システム105により確定されるプロクシオブジェクト100は実オブジェクト103と、実オブジェクト103をいかにしてアクセスし且つ操作するか、たとえば、その方法をいかにして呼び出し、その特性をいかにして設定し、例外をいかにして処理するかを記述するソフトウェア記述104に対するポインタを含む。プロセス101がプロクシオブジェクト100を操作するとき、その操作はシステムによりインタセプトされて、実オブジェクト103へと送られる。システムはソフトウェア記述104中の命令に続いて操作を進め、その結果、実オブジェクト103の対応する操作が実行される。システムは、どのオブジェクトシステムがその命令を作成したかにかかわらず、ソフトウェア記述104中の命令に従うことができるので、オブジェクトシステムのインタオペラビリティに対する非( $N^2 - N$ )アプローチとなる。

図2を参照すると、本発明の好ましい一実施例に従ったシステムの簡略化した

アーキテクチャが示されている。後述するように、出願人は広範囲にわたるオブジェクトシステムにおいて広範囲にわたるハードウェアプラットフォームについてこの実施例を実施し、申し分のない結果を得た。図中、三角形の記号110は、その右側にある項目がサブクラスであることを指示する。システムは次のような9つのフレームワークを含む。

ロケーション・列挙フレームワーク (Enumeration Frame work) 1 :

オブジェクトシステムに特有の又は均一なネームスペースにおいてオブジェクトの場所を動的に規定する。このフレームワークには、場所を指定されたオブジェクトの特性を確定する能力が含まれている。

クラス記述フレームワーク 2 :

オブジェクトクラス又はインスタンスの特性を記述する。

タイプ記述フレームウェア 3 :

意味タイプと言語タイプ双方を記述し且つ変換する。

フォワーディング・エンジン・フレームワーク 4 :

プロキシオブジェクトの操作を実外部オブジェクトへ送り出すためにソフトウェア記述を「実行 (execute)」する。

誤り・例外処理フレームワーク 5 :

誤り及び例外を処理する。

ライフサイクル管理フレームワーク 6 :

オブジェクトのライフサイクルを生成し、コピーし、破壊し、管理する。

オブジェクトエクスポートフレームワーク 7 :

オブジェクト定義を「エクスポート (export)」する。すなわち、1つのオブジェクトシステムにおけるオブジェクトを他のオブジェクトシステムで一時的に又は永久にネイティブオブジェクトクラスとは区別不可能なオブジェクトクラスとして見えるようにする。

加えて、1つのオブジェクトシステムでのみ見いだされる特徴を全てのオブジェクトシステムで支援できるように、どのオブジェクトクラスの記述にも余分な情報を追加するためのミキシン支援フレームワーク 8 がある。また、オブジェク

トシステムアダプタ (OSA) 10をロードし、アンロードし、管理するためのOSAレジストリ・フレームワーク9もある。OSA10は、システムにより提供される前述のフレームワークの各々のデフォルト・インプリメンテーションに対する拡張を一体にパッケージするライブラリである。OSAは、追加されると、対応するOSAがロードされている他の全てのオブジェクトシステムの間に完全な両方向インタオペラビリティを成立させる。

更に詳細なアーキテクチャについては、実施例の詳細な説明の終わりに添付したVisual Edge Software 社のClass Registry Functional Specification (ページ141からページ143)を参照のこと。上記のアーキテクチャをここで説明する実施例の有効性に影響を及ぼすことなく再編成し得ることは当業者には理解されるであろう。

図3は、OSAレジストリフレームワーク9と、ロードされるいくつかのOSAとを示す。この構成においては、システムはそれらのOSAに対応する4つのオブジェクトシステムの間にインタオペラビリティを提供している。

本発明の特定の実施例はC++で書き表されており、マイクロソフトのウインドウズ (Windows3.1) 及びWindows NT, IBMのOS/2 及びAIX、サンマイクロシステムズ (Sun Microsystems) のSunOS及びSolaris, 並びにヒューレットパカード (Hewlett Packard) のHP-UXで実行される。オブジェクトシステムアダプタはマイクロソフトのOLE Automation, IBMのSOM及びDSOM、マイクロソフトのCOMに対して実行されており、マイクロソフトのVisual Basic VBXオブジェクトシステムに対応する。さらに、OSAは解釈言語環境に合わせて実行されている。現時点では、この実施例は対応するOSAを伴うオブジェクトシステムにおいて使用されるべき純粋C++でインプリメントされたオブジェクトをイネーブルするための支援を実行する。同様に、この実施例は現時点では対応するOSAを伴うオブジェクトシステムにおいて使用されるべきCで書き込まれた (すなわち、どのオブジェクトシステムにも書き込まれていない) ソフトウェアをイネーブルするための支援を実行する。

以下の各章は上記のフレームワークをそれぞれ説明し、以下の各章の番号は図

2の項目番号に相当する。

1) ロケーション・列挙フレームワーク：

オブジェクトシステムに特有の又は均一なネームスペースにおけるオブジェクトの場所を動的に指定するためのフレームワーク。

ロケーション・列挙フレームワークは、図2bに示すように、「アダプタネームスペース(Adapter NameSpace)」111と呼ばれるオブジェクトシステムに特有のネームスペースサブフレームワークと、「ビューネームスペース(View Name Space)」112と呼ばれるオブジェクトシステムとは無関係のネームスペースサブフレームワークの2つの主要サブフレームワークを有する。これらが合わさると、新たなオブジェクトシステムについてフレームワークにおける支援を追加する際に、アダプタネームスペースをサブクラス化するという線形（すなわち、非( $N^2 - N$ ))動作を実行するだけで良い。

ロケーション・列挙フレームワークは、OSAにより使用されるか、又はオーバーライドされ（オブジェクト指向の意味で）、個々のオブジェクトシステムに特有な能力と置き換えられる一連の総称(generic)能力を提供する。すなわち、ア

ダプタネームスペース111は、オブジェクトシステムの中で1つの特定のクラス、インスタンス、機能、タイプ、例外、サブネームスペース、あるいはクラス、インスタンス、機能、タイプ、例外、サブネームスペースの完全なリスト、もしくはその何らかの組み合わせの場所を指定するための能力の探索を実行する。この探索プロセスの間に、正規表現整合などの周知の数多くの探索技法を適用できることは当業者には明白であろう。従って、オブジェクト指向原理を使用すれば、単一のインタフェース、すなわち、アダプタ・ネームスペース・インタフェースをシステムのその他の部分により利用して、どのような特定のオブジェクトシステムでも探索し、それが提供する情報の階層を確定して、戻せる。

アダプタネームスペースの項目は、システムの実行時又は特定のOSAのローディング時に列挙できる。しかしながら、多くの場合、アダプタネームスペースは、ユーザがブラウジング操作を実行するか、システムが特定のクラスのオブジェクトを構成する必要が生じたか、又はシステムが別のオブジェクトシステムで

も見えるクラスを作成する必要が生じたときのように、要求が発生するまでは内容を列挙しない。

アダプタネームスペースは、システムにより提供される総称メカニズムを使用するか、又はオブジェクト特有メカニズムを供給することによって内容を列挙する。システムが提供する総称メカニズムには、静的メカニズム、動的メカニズム又はデータベースメカニズムが含まれる。静的メカニズムの場合、システムのユーザはシステムAPIを呼び出して、ユーザの情報を登録する。動的メカニズムの場合には、システムはオブジェクト自体（又は言語インタプリタの記号テーブルのような、そのシステムを利用しているソフトウェア）に問い合わせ、情報を確定する。データベースメカニズムの場合には、システムはファイル又はレポジトリ(repository)を読み取って、情報を確定する。レポジトリを読み取る場合の例としてはCORBAインタフェースレポジトリから情報を問い合わせること、COMタイプライブラリを読み取ること、「ヘッダファイル」（たとえば、C++ヘッダファイル又はCORBA IDLファイル）を読み取ること、又は「ダイナミック・リンクライブラリ」の記号テーブルを読み取ることが挙げられるであろう。通常は、各OSAは総称メカニズムのいくつかの能力をオーバーライドし

なければならないであろう。

第2のサブフレームワークであるビューネームスペース112は、クラス、インスタンス、タイプなどの情報を編成するためのオブジェクトシステム独立メカニズムを提供する。ビューネームスペースは非巡回グラフで編成されても良い。また、ビューネームスペースをサブクラス化して、名前付きオブジェクト（すなわち、クラス、インスタンス、タイプなど）のフラットリストのような特定の能力を提供しても良い。アダプタネームスペースで列挙される項目は、いずれも、OSAのネームスペース全体を含めて、ビューネームスペースの中に配置できる。ファイルシステムにおいては通常見られることであるが、エイリアスを提供すると共に、循環性を認識する。このように、システムのユーザはオブジェクトシステムとは全く無関係に完全に情報の場所を指定し且つ情報を編成する手段を得る。

## 2) クラス記述フレームワーク (Class Description Framework) :

オブジェクトクラス又はインスタンスの特性を記述するためのフレームワーク  
アダプタネームスペース及びビューネームスペースには、クラスを記述するための情報が含まれている。クラス記述フレームワークは、この能力をイネーブルにすると共に、O S Aが組み込み機能性をオーバーライドできるようにするために提供される。クラス記述フレームワークは、クラス、インスタンス、特性、機能（方法を含む）、引数及び例外を記述する一連のクラスから構成されている。使用される対応クラスは、それぞれ、V C l a s s D a t a, V I n s t a n c e D a t a, V P r o p D a t a, V F u n c t i o n D a t a, V A r g u m e n t D a t a及びV E x c e p t i o n D a t aと名付けられている。以下に説明するように、タイプを記述するための追加フレームワークであるタイプ記述フレームワークも利用される。上記のクラスの各々は、それが表わすオブジェクトの名前、そのタイプ、その所有者（たとえば、クラス特性を記述している場合には、所有者はそのクラスということになるであろう）、それを管理するオブジェクトシステム並びに「利用 (usage) コード」（後述する）について求めることができる。

クラスを記述するために使用されるクラス記述フレームワーク中のクラスは、

（特性又は方法を記述する場合とは異なり）V C l a s s D a t aと呼ばれる。V C l a s s D a t aは、静的データ構造、関係リストデータ構造などを構築するのではなく、方法呼び出しの結果を戻すことによって、システムのその他の部分に対してV C l a s s D a t aが表わすクラスの記述を提供する。たとえば、特性のリストを戻すクラス方法 (class methods) を提供する。これにより、O S AはV C l a s s D a t aをサブクラス化し、その方法をオーバーライドして、システムのその他の部分に対して依然として単一のA P Iを提供しつつ、オブジェクトシステム特有インプリメンテーションを使用してクラス記述を行うことができる。

V C l a s s D a t aは、それが記述するクラスについて、ベースクラス（V C l a s s D a t aのリストとして）；コンストラクタ、デプリケータ及びデ

ストラクタ (VFunctionDataとして) ; クラスの方法 (VFunctionDataのリストとして) ; クラスの特性 (VPropDataとして) ; クラスの例外 (VExceptionDataのリストとして) を戻すための方法 ; 並びにクラスの名前付きインスタンスを戻すための方法を有する。

機能とクラス方法を記述するクラス記述フレームワーク中のクラスをVFunctionDataと呼ぶ。VFunctionDataクラスは、それが記述する機能について、引数 (VArgDataとして) 、機能がスロー (throw) できる例外 (VExceptionDataのリストとして) 及び機能に関する呼び出し可能エントリポイント (又は方法) を戻すための方法を有する。

クラス特性を記述するクラス記述フレームワーク中のクラスをVPropDataと呼ぶ。VPropDataクラスは、それが記述する特性について、特性をセットするための方法並びに特性を獲得するための方法 (共にVFunctionDataとして) を戻すための方法を有する。

同様に、機能引数を記述するクラス記述フレームワーク中のクラスであるVArgDataは、それが記述する引数について名前、タイプ及び利用コードを戻すための方法を有する。インスタンスを参照するクラスであるVInstanceDataは、その値 (すなわち、実際のインスタンス) を獲得し、且つセットするための方法を有する。例外を記述するクラスに関しては、以下の誤り・例外

処理の章で説明する。

上記の情報は、通常、ネームスペースが列挙されるときにOSAにより構築されるが、システムのユーザによって直接に構築されても良い。OSAがその内容を列挙するときにはどの能力もオーバーライドしない場合、通常、OSAは、情報を検索する方法を有するのに加えて情報をセットするための相応する方法を有する上記のクラスの各々のサブクラスを使用するであろう。そのため、オブジェクトシステムの内容を確定するとき、OSAは上記のクラスの必要なインスタンスを構築し、それらを「連結」するための方法と呼び出すであろう。たとえば、OSAはそれらの方法を支援するVClassDataのサブクラスを構築し、次に、そのクラスの特性に対応するVPropDataを構築し、更に、VClass

sDataの中の特徴のリストをセットするための方法と呼び出すであろう。図4は、OSAが1つの方法12と、1つの特性13とを伴うDSOMクラス11の記述を構築した結果を示す。この方法は1つの引数14及びショート(short)15を取り出し、ロング(long)16を戻す。特性13はショートのタイプ17であり、ショート引数19及び20をとる値18をセットする機能を有し、ボイド(Void)21を戻す。特性は、ショート23を戻す値22を獲得する機能をも有する。図4に示すように、情報をツリーとして編成する必要がないことは当業者には明白であろう。

### 3) タイプ記述フレームワーク (Type Description Framework) :

意味タイプと言語タイプの双方を記述するためのタイプ管理フレームワーク

クラス記述フレームワークと同様に、タイプ記述フレームワークは、意味タイプと言語タイプの双方を含むタイプを記述するためのものである。タイプ記述フレームワークは、VTypeDataと呼ばれるタイプを記述するためのベースクラスと、整数、フロート、ダブル、バイト、符号なしロングなどの基本タイプを表わすVcrFundamentalと呼ばれるVTypeDataの1つのサブクラスとを提供する。更に、構造 (VcrStruct), ユニオン (VcrUnion), ポインタ (VcrPointer) 及びファンクションポインタ (VcrFunctionPointer), オブジェクト参照 (VcrObjectRef), アレイ (VcrArray) 及びストリング (VcrStr

ing) を含むシーケンス (VcrSequence)、エニユムス (enums) (VcrEnum) などの複合タイプをVTypeDataのこれらのサブクラスを使用して記述することができる。また、任意に複合タイプを生成するために、タイプを入れ子型にすることも可能である。図5は、2つの項目26及び27を含む構造25に対するポインタ24であるタイプの記述を示す。第1の項目はショート28であり、第2の項目はロング29である。

### 4) フォワーディング・エンジン・フレームワーク (Forwarding Engine Framework) :

これらの記述は、プロキシオブジェクトの操作をを実外部オブジェクトへ送り

出すことをどのようにして可能にするのであろうか？

クラス記述及びタイプ記述という上記の要素を組み合わせ、1つのクラスを完全に記述する。この方法の鍵となる2つの面は、意味論的にそれらの要素が有意味である否かに関して要素にタグを付すことと、記述の中に意味情報と、非意味情報の双方を含めることである。その結果、この記述方法は基本呼び出しメカニズムと、言語呼び出しコンベンションとを完全に記述することができる。たとえば、図6 aは、CORBA呼び出しコンベンションに従った単純な方法呼び出しに関する宣言を示す。方法35は、オブジェクト30と、環境ポインタ31（全てのCORBA方法呼び出しの特性、環境ポインタは誤りを戻すためのメカニズムとして最も一般的に使用される）と、X32及びY33と名づけられた引数という4つの引数をとる。この方法はポイド34を戻す。図6 bは、この方法の対応する記述を示す。

図7 aは、異なる呼び出しコンベンションを使用する別のオブジェクトシステムにおける同じ方法を示す。この宣言においては、ダイナミック・インボケーション・インタフェース（DII）を使用して、方法を読み出す。第1の引数36はオブジェクトである。第2の引数は方法37の識別子である。次は引数38の数のカウントである。最後に、これら引数はアレイ39にパックされる。DIIは誤りポインタ40を戻す。図7 bは、この呼び出しの対応する記述を示す。尚、いずれの場合にも、オブジェクトX及びYのみが意味論的に有意味（SM）のものとしてタグを付される。記述される他の面はその方法呼び出しをいかにして

実行すべきかのアーティファクトであるにすぎない。

要するに、1つのオブジェクトから別のオブジェクトへ方法呼び出しを送り出すために、フォワーディングエンジンは実行された方法呼び出しの記述をウォークダウン(walk down)し、呼び出しスタックから意味論的に有意味の全ての情報を引き出す。次に、フォワーディングエンジンはそれが実行しようとしている方法呼び出しの記述をウォークダウンし、記述された全ての情報を「コールスタック(call stack)」に挿入し（コンピュータシステムの各プロセッサは機能呼び出しを実行する前に引数をどの場所に挿入すべきかに関して特定の規則を有する、

そのような場所を一般には「コールスタック」と呼ぶ)、意味論的に有意味の情報を正しい位置に転送する(また、必要に応じて、タイプを変換する)。次に、実際に方法呼び出すための正しい機能呼び出す。方法が戻ったならば、同様にその結果を元の方法コールスタックへ転送して、戻す。

図 7 c には、以下では外部オブジェクトシステムと呼ばれる図 7 のオブジェクトシステムで実行されるオブジェクトが、以下ではネイティブオブジェクトシステムと呼ばれる図 6 のオブジェクトシステムに書き込まれたソフトウェアにより使用されている場合に、図 6 の方法 3 5 を呼び出したときに実行されると思われるステップが示されている。

まず、OSA がプロキシを構築した方式に基づいて、OSA はそのプロキシを操作したソフトウェアから制御の流れ 7 0 1 を受け取る。次に、OSA が構築したプロキシのレイアウトに基づいて、ネイティブオブジェクトシステムの OSA は実オブジェクトと、実オブジェクト及びプロキシオブジェクトの双方に関する `VFunctionData` とを検索する (7 0 2)。これらはフォワーディングエンジンフレームワークに渡される。代表的なプロキシオブジェクトのレイアウトを図 1 0 に示すと共に、以下の 7 章で説明する。

フォワーディングエンジンフレームワークの第 1 の主要ステップはプロキシの `VFunctionData` をトラバースし、`VFunctionData` を使用して、発生したばかりの方法呼び出しから意味論的に有意味の全ての情報を検索する (7 0 3)。フォワーディングエンジンフレームワークはその引数のリスト(すなわち、`VcrArgument` のリスト)を尋ねる。引数ごとに、その

引数が意味論的に有意味であるか否かを判定する。引数自体が意味論的に有意味でなく、その引数が複合タイプである場合、引数はそれ自体の中に意味論的に有意味の情報をまだ含んでいると考えられる。従って、エンジンは引数の中に意味論的に有意味の情報が存在しないことを確認するために、引数の内容を繰り返しトラバースしなければならない。

意味論的に有意味でない情報は放棄され、意味論的に有意味の情報は記憶される。この例では、第 1 の引数 3 0 は方法が呼び出されたオブジェクトである。こ

れは意味論的に有意味であるので、記憶される（ただし、プロキシオブジェクトに対するこの識別子はネイティブOS Aにより既に検索されているので、最適化という意味では記憶する必要はない）。

次の引数31は意味論的に有意味ではないので、無視される。第3及び第4の引数32及び33は、それぞれ、意味論的に有意味であるので、検索、記憶される。たとえば、インテル486マイクロプロセッサで実行される機能呼び出しから情報を検索するための方法の詳細な説明については、本明細書に参考として取り入れられているIntel486 Microprocessor Family Programmer's Reference Manual（1992年）を参照。

元の方法呼び出しから意味論的に有意味の全ての情報を検索したならば、フォワーディングエンジンが実行する次の主要ステップは、「コールスタック」に適切な引数を配置することにより、実オブジェクトに関して方法呼び出しを構築し始める（704）。この例では、フォワーディングエンジンは外部オブジェクト（すなわち、実オブジェクト）の方法をいかにして呼び出すかを表わす、図7に示すVFunctionDataをトラバースし、引数のリストを検索するであろう。

第1の引数36は意味論的に有意味であるので、これをコールスタックに挿入しなければならない。フォワーディングエンジンは、整合する情報を見出すために、元の方法呼び出しから検索した情報のリストをトラバースする。この場合にはオブジェクト。元の呼び出しからのオブジェクトを外部呼び出しより要求されるタイプに変換し、スタックに挿入しなければならない。フォワーディングエンジンは要求されるオブジェクトが実オブジェクトであるか否かを確認する（この

場合には実オブジェクトである）という最適化を含み、従って、その実オブジェクトをコールスタックに挿入する。第2の引数37は、フォワーディングエンジンがコールスタックに挿入するストリング定数「MethodName」（すなわち、実方法の名前）である。同様に、第3の引数38は値が2である定数ショートであるので、コールスタックには2が挿入される。

第4の引数39は複合タイプであるので、フォワーディングエンジンはその要

素をトラバースする。フォワーディングエンジンは2つのロング整数を含むことができるアレイを生成する。次に、フォワーディングエンジンは第1のエントリを元の呼び出しから検索された情報と整合し、第1の位置にXを挿入する（ロング整数として、タイプ変換は不要）。第2の引数Yについても同じことを実行する。これが完了したならば、フォワーディングエンジンはアレイをコールスタックに挿入する。

フォワーディングエンジンフレームワークにより実行される第3の主要ステップは、実際の方法を呼び出す（705）。フォワーディングエンジンは呼び出すべき機能のアドレスをVFunctionData41に要求する。この例では、フォワーディングエンジンはDII機能のアドレスを戻す。次に、フォワーディングエンジンは実際に機能呼び出すためにポインタをデリファレンスする。

フォワーディングエンジンにより次に実行されるステップは、戻り値を処理する（706）。フォワーディングエンジンはコールスタックから戻り値を検索する。その戻りタイプが、元の方法呼び出しのVFunctionDataに要求される。元の方法呼び出しはボイドを戻すので、「戻りスタック」に情報を挿入する必要はない（「戻りスタック」という用語は、一般に、機能呼び出しの結果が記憶されるコンピュータプロセッサ特有の1つ又は複数の記憶場所を表わす）。

元の方法呼び出しから戻る前の最終ステップは誤り及び例外の処理である。この例では、外部方法は誤り情報に対するポインタを戻す（707）。VFunctionDataの元来のトラバースは、環境ポインタを誤り情報を含むものとして識別していた（環境ポインタは誤り情報を転送するためのネイティブオブジ

ェクトシステムの標準メカニズムである）。誤り及び例外処理フレームワークは、例外が起こったか否かを判定し、情報を外部オブジェクトシステムにおいて実オブジェクトに対する方法呼び出しにより提供された形態からネイティブオブジェクトシステムに適する形態に変換するタスクを委託される。この例では、必要な変換が起こり、制御はフォワーディングエンジンに戻され、最終的には、元々、方法35を呼び出したプロセスに戻される。

尚、フォワーディングエンジンが実行する一連のステップをソースコードとして自動的に生成し、希望に応じて、実行のためにコンパイルすることが可能である。同様に、以上の説明は方法呼び出しに関連してなされていたが、あらゆる操作に適用されることに注意する。更に、特に説明のない限り、操作中であるオブジェクトがオブジェクトシステムから出たものである必要はない。たとえば、変更なしのC言語ソフトウェアがどのようなオブジェクトシステムでも実行されるように見せることができる。

OSAは、システムのフォワーディングエンジンフレームワークコードに方法の送り出しを自動的に処理させるか否か、あるいはカスタムフォワーディング方式を提供するためにフレームワークをサブクラス化するか否かについて選択できる。場合によっては、OSAは実行しようとしている方法呼び出し（実行されたばかりの、送り出さなければならない方法呼び出しではない）の記述を見て、コールスタックからどの意味情報を除去しなければならないかを判定することができる。OSAは呼び出しがちょうど実行されたオブジェクトシステムに対応するため、OSAライタはコールスタックから情報をどのようにして除去すべきかをOSAにコード化することができる。すなわち、OSAはそれ独自の方法の一般記述をウォークダウンする必要がない。

同様に、実方法（すなわち、外部オブジェクトシステムの方法）に関してVFunctionDataを構築したOSAは、実呼び出しを実行するために必要であるとわかったどのような手段でも利用するように構築することができる。たとえば、OSAは、更にカスタム処理を実行するためにOSAへ「コールバック」するようにVFunctionDataを構成することができる。

上記のクラスの記述は、ある特定のオブジェクトシステムに特有のものではない。クラスを記述するためにどのオブジェクトシステムによっても生成可能であり、いずれかのオブジェクトシステムによるオブジェクトの操作をいずれかのオブジェクトシステムへ送り出すために、どのようなOSAによってもトラバースでき、「実行」できる。そのため、オブジェクト操作を送り出す方法は（Nの2乗－N）方式ではない。

オブジェクト自体の記述のためにオブジェクトが問い合わせられる場合にも類似の方法が使用される。ネイティブオブジェクトの記述は、プロキシによりトリガされて、フォワーディングエンジン又は O S A のいずれかによってトラバースされ、意味情報はその要求を発したオブジェクトシステムが要求するフォーマットに変換される。そのような情報のフォーマッティングは O S A の責務の 1 つである。

前述のように、何らかのタイプを含むストリング及び変数などの意味タイプ並びに言語タイプを含むタイプは、多くの場合、オブジェクトシステム相互間で変換されなければならない。システムは、2つのオブジェクトシステムの間で意味タイプ及び言語タイプを非 ( $N$  の 2 乗  $- N$ ) 方式で変換することができるオブジェクト指向タイプシステムであるタイプ記述フレームワークを提供する。フォワーディングエンジンは、呼び出されたばかりの方法の記述の中のタイプ情報を利用することにより、コールスタックから情報を引き出す。呼び出されようとしている方法の記述をウォークするとき、フォワーディングエンジンは同様に予期されるタイプをも知る。元来のタイプと予期されるタイプの双方を知った後、タイプ記述フレームワークは変換を実行する。

この特定の実施例においては、タイプ変換に際して、オブジェクト指向タイプキャスト及び中性タイプ表現という 2 つのメカニズムが提供される。オブジェクト指向タイプキャストでは、タイプ記述フレームワークは全てが同一のタイプのサブクラスであるタイプの間で変換を実行することができる。これは、それぞれのサブクラスがインプリメントする一組の方法を提供するスーパークラス `VTypeData` によって実行される。これらの方法はインスタンスを構築するために必要な情報を戻す。この情報はサブクラスタイプのいずれをも十分に生成できる。従って、変換時には、スタックからちょうど引き出されたタイ

プのインスタンスについてこれらの方法呼び出す。その結果は、スタックに挿入されようとしているタイプのインスタンスを生成するために必要な情報を提供する。

中性タイプ表現では、タイプはそれら自体を 1 つ又は複数の中性タイプに変換

する方式、並びに中性タイプからそれら自体に変換する方式を知る。たとえば、`String`のサブクラスとしての`COM_BStr`のような`String`タイプはそれ自体を`ASCIIString`又は`UnicodeString`に変換する方法並びに`ASCIIString`又は`UnicodeString`から`BStr`に変換する方法を有していると考えられる。従って、いずれかの`String`タイプから他の何らかの`String`タイプに変換するときには、第1のタイプはそれ自体を`ASCII`又は`Unicode`に変換し、第2のタイプは`ASCII`又は`Unicode`からそれ自体に変換する。尚、タイプはどちらの変換が最も効率が良いかを承認することができ、変換は必要なときにのみ実行される。更に、以上の説明は、タイプ変換の最適化が望まれる特定のオブジェクトシステム（及びタイプ）相互間での直接タイプ変換メカニズムの追加を排除するものではないことが当業者には認められるであろう。

タイプ記述フレームワークは自動プロキシ構築をトリガする目的でも使用される。すなわち、オブジェクトタイプを渡すとき、オブジェクトを呼び出されるべきオブジェクトシステムのオブジェクトに「変換」しなければならない。そのため、プロキシを構築する必要がある。プロキシ構築に関わる詳細なステップについては、6) 章及び7) 章で説明する。

5) 誤り及び例外処理フレームワーク (Error and Exception Handling Framework) :

誤り及び例外を処理するためのフレームワーク

誤り及び例外処理フレームワークは、タイプ変換システムに大きく依存している。どの`VFunctionData`についても、その一部として`VExceptionData`を生成する。何らかの例外が起こったときには、タイプ変換システムはその例外を1つのオブジェクトシステムにおけるタイプから別のオブジェクトシステムの対応するタイプに変換する。加えて、タイプ変換メカニズム

は`VExceptionData`の方法を介して、外部オブジェクトシステムの例外処理システムをトリガする働きをする。

6) ライフサイクル管理フレームワーク (Lifecycle Management Framework) :

オブジェクトのライフサイクルを生成し、コピーし、破壊し、管理するためのフレームワーク。

7) オブジェクト・エクスポート・フレームワーク (Object Exporting Framework) :

オブジェクト定義をエクスポートするためのフレームワーク。

オブジェクトのライフサイクルはオブジェクトシステムごとに異なるが、代表的なオブジェクトシステムにおける1つのオブジェクトのライフサイクルを図8に示す。プロセスは、ユーザがオブジェクトのインスタンスを生成しようとしたときに始まる(42)。まず、オブジェクトシステムは、オブジェクトを生成するためのファクトリが利用可能であるか否かを判定する(43)。ファクトリが利用可能でないならば、オブジェクトを生成するための呼び出しの中で指定される通りに、オブジェクトシステムはサーバをスタートさせようとする(44)。オブジェクトシステムはサーバアプリケーションをスタートし(又は適切な動的関係ライブラリをロードし)、サーバアプリケーションへ実行を移行する(45)。サーバは初期設定を実行し、それが支援する全てのクラスに対してファクトリを生成する(46)。次に、場合によっては「ファクトリ露出」と呼ばれるプロセスの中で、サーバはそのファクトリをオブジェクトシステムと共に登録する(47)。そのプロセスが完了すると、サーバは制御をオブジェクトシステムに戻す(48)。そこで、オブジェクトシステムは所望のクラスのインスタンスを生成するために適切なファクトリの方法を呼び出す(49)。ファクトリはオブジェクトのインスタンスを生成し(50)、最後に、そのインスタンスがユーザに戻される(51)。オブジェクトが利用される間に、オブジェクトに対する追加参照が生じる場合もある(52)。オブジェクトは、それ自身に対する参照を有するいずれかの相手により破壊される場合もある(53)。オブジェクトを破壊するためにオブジェクトへの参照を使用するたびに、サーバはオブジェクトクリーンアップが必要であるか否か(たとえば、オブジェクトが使用しているメモリを割り当て解除すべきか否か)を判定する(54)。典型的には、これは最終参照を破壊するときに起こる。

システムは、インタオペラビリティをトランスペアレントに提供する。その結果、システムは、オブジェクトシステムに対して、オブジェクトシステムに合わせて特別に設計されたどのサーバも従うと考えられる標準ライフサイクルプロトコルに従う標準オブジェクトサーバとして現れなければならない。従って、システム、更に特定すれば、要求が起こったオブジェクトシステムのOSAは、項目5及び6、項目8.5を含む項目3について責務を負い、10及び12の間に、オブジェクトの利用／操作が適正に且つトランスペアレントに実オブジェクトへ送り出されるように保証している。ところが、8.5の間には、OSAは別のOSA、すなわち、実オブジェクトが生成されたオブジェクトシステムのOSAと共に機能して、実オブジェクトのプロキシを生成する。

システムは上記のステップを実行するためのフレームワーク、すなわち、オブジェクト・エクスポート・フレームワーク (Object Exporting Framework) を提供する。このシステムを含むどのアプリケーション／サーバも44になりうるであろう。図9に示すように、ステップ46を実行するときに、アプリケーションはそれ自体を初期設定し(55)、次にシステムを初期設定する(56)。別のオブジェクトシステムからのクラスごとに、アプリケーション／サーバはこのオブジェクトシステムにおいて利用可能にすることを望み、要求側のオブジェクトシステムのOSAでExposeFactory方法呼び出す(57)。そのようなクラスのリストは、持続ストア又はファイルから確定されるか、あるいは動的に確定されて、アプリケーション／サーバでハードコード化されてもよい。ExposeFactory方法は露出させるべきクラスのVCClassDataと共に呼び出され、ネイティブオブジェクトシステムから実ファクトリオブジェクトを戻す。これは必要に応じて新たなファクトリを構築するか(58)、又は先に生成されていたファクトリを戻す(59)。また、その方法はそのファクトリをネイティブオブジェクトシステムと共に登録する(60)。

クラスを露出させることのもう1つの面は、それらをネイティブオブジェクトシステムの「レポジトリ(repository)」に登録することである。これは外部オブジェクトシステムに適する「言語」(CORBAシステムにおけるインタフェー

ス定義言語、すなわち、IDL、COMのタイプライブラリ、又はC++ヘッダファイル)でインタフェース記述を生成することを含んでいても良い。これには、OSAがオブジェクトシステムのレポジトリと共にクラスを登録するためにAPIを使用することも要求されるであろう。

オブジェクトインスタンスを生成するためにオブジェクトシステムがファクトリの方法を呼び出すとき、外部オブジェクトシステムにおいて実インスタンスを生成し、次にネイティブオブジェクトシステム(すなわち、ファクトリの設計の元になったオブジェクトシステム)に戻されるべきプロキシを生成するのは、ファクトリのジョブである。ファクトリを構築するとき、OSAはそのオブジェクトシステムのオブジェクトに関わるVC l a s s D a t aを構築する。このVC l a s s D a t aと、実オブジェクト(ExposeFactory方法呼び出しでOSAが提供されたオブジェクト)とはファクトリオブジェクトと共に記憶される。オブジェクトを生成するためのファクトリオブジェクトの方法がオブジェクトシステムによって呼び出されると、ファクトリオブジェクトはフォワーディングエンジンを使用して、作成呼び出しをそのオブジェクトシステムから(生成したVC l a s s D a t aを使用して)ネイティブオブジェクトシステムへ送り出す。実オブジェクトが戻されると、ファクトリオブジェクトは外部オブジェクトシステムに関わるOSA(更に厳密に言えば、OSAで提供されるオブジェクト・エクスポート・フレームワークのサブクラス)においてAcquireProxy方法と呼び出すことによりプロキシオブジェクトを構築する。

AcquireProxy方法は実オブジェクトのVC l a s s D a t aと、オブジェクトのインスタンスとを渡される。方法はネイティブオブジェクトシステムでプロキシオブジェクトを構築し、そのオブジェクトと、そのOSAにより要求される全ての情報に対するポインタを関連づける。プロキシオブジェクトのレイアウトはOSAまでであるが、代表的なプロキシオブジェクトを図10に示す。プロキシオブジェクト61は、その方法テーブル62(C++ではv t a b l eと呼ばれる)に対するポインタと、メタデータオブジェクト63に対するポインタとから構成されている。メタデータオブジェクトは、方法テーブル64に

対するポインタと、ネイティブオブジェクト65のVClassDataに対するポインタと、ネイティブオブジェクト識別子66と、プロキシオブジェクト67のVClassDataに対するポインタと、経路短縮情報68とから構成されている。

ネイティブオブジェクトシステムでクラスのインスタンスが初めて生成される時、オブジェクトシステムのAcquireProxy方法は一般的には方法テーブルを生成しなければならない。方法テーブルを動的に生成する方式と、OSAがどの方法が呼び出されたかを検索するのに十分な情報を各方法と関連させる方式は当業者には理解されるであろう。方法テーブルを介してオブジェクトシステムにより方法が呼び出されたとき、通常は次のステップが実行される：まず呼び出されたばかりのプロキシオブジェクトの方法に関わるVFunctionDataを検索する。次に、プロキシオブジェクトのメタデータからネイティブオブジェクトのVClassDataを検索し、今度はネイティブオブジェクトの対応する方法に関わるVFunctionDataを検索する。これらは方法呼び出しの送り出しのためにフォーワーディングエンジンに渡される。

プロキシオブジェクトの構成は42におけるようにユーザにより、又はタイプ変換システムによりトリガできる。タイプ変換によってトリガされる場合には、AcquireProxyを呼び出すだけで良い。

AcquireProxy方法を呼び出すとき、新たなプロキシオブジェクトを生成するのに先立って、ネイティブオブジェクトのプロキシオブジェクトがこのオブジェクトシステムの中に既に存在しているか否かを検査しなければならない。システムにより提供される能力を利用して、方法はオブジェクトのキャッシュを検査する。プロキシオブジェクトが存在しているならば、それを戻し、そうでなければ、システムは経路短縮が要求されるか否かを検査する。オブジェクトシステムのインタオペラビリティを提供するために複数のシステムが協働しているとき、プロキシを構築すべきオブジェクトそれ自体がプロキシオブジェクトである可能性がある。経路短縮(path shortening)とは、そのプロキシに関してプロキシを生成せず、その代わりに「オブジェクトの経路を短縮し」、実オブジェクトに関してプロキシを生成する動作を表わす。たとえば、本明細書の中にも参

考として取り入れられている「SSP Chains: Robust, Distributed References Supporting Acyclic Garbage Collection, Shapiro, Dickman及びPlain Fosse著, Symposium on Principles of Distributed Computing (1992年8月)」などの文献に、数多くの経路短縮アルゴリズムが記載されている。

#### 8) ミキシン支援フレームワーク (Mixin Support Framework) :

オブジェクトクラスの記述に余分の情報を追加するためのフレームワークであり、1つのオブジェクトシステムでのみ見出された特徴を全てのオブジェクトシステムで支援できるようにする。

多くの場合、1つのオブジェクトシステムは、別のオブジェクトシステムでは利用できない特徴を提供する。更に、オブジェクトシステムはそのオブジェクトがいくつかの特徴を提供することを期待する場合が多い。システムは、ミキシンと呼ばれるそのようなインタオペラチリティの問題を処理するための汎用メカニズムを提供する。ミキシンは、単に、システム中の他の何らかのオブジェクトが提供できる機能性を拡張するためにそのオブジェクトと関連づけることができるオブジェクトであるにすぎない。ミキシンはネイティブオブジェクトが提供しないか、あるいはネイティブオブジェクトの方法のいずれか又は全てをオーバーライドする方法を提供できる。全てのフレームワークはミキシン支援フレームワークからサブクラス化されているので、ミキシンオブジェクトはシステムから提供されるどのクラスのどのインスタンスにも追加できる。

フォワーディングエンジンがネイティブオブジェクトの方法を求めて `VFunctionData` を検索するために `VClassData` をウォークダウンするとき、まず、フォワーディングエンジンは方法を提供するオブジェクトと関連するミキシンオブジェクトが存在するか否かを判定する。存在するのであれば、エンジンは（ネイティブオブジェクトについて方法が存在するとしても）ネイティブオブジェクトの方法ではなく、ミキシンの方法呼び出す。ミキシンオブジェクトの方法をトリガして、ネイティブ方法呼び出しの前又は後に、ネイティブオブジェクトの能力を増補することも可能である。システムにクラスレベル又はインスタンスレベルで知られているオブジェクトとミキシンオブジェクトを関連づけても良い。更に、ミキシンオブジェクトはオブジェクトのサブクラスが列挙

されるにつれて継承される。ミキシン支援メカニズムを介して、ネイティブオブジェクトはその能力が増補されたことを認識する必要はない。

9) OSAレジストリフレームワーク (OSA Registry Framework) :

オブジェクトシステムアダプタをロードし、アンロードし、管理するためのフレームワーク。

先に説明したように、オブジェクトシステムアダプタは、上記のフレームワークのシステムのデフォルトインプリメンテーションの拡張を共にパッケージするライブラリである。OSAはそれが対応するオブジェクトシステムと、OSAを伴う他の全てのオブジェクトシステムとの間の両方向インタオペラビリティを完全に支援するために必要なあらゆるものをパッケージしている。システムはOSAレジストリフレームワークと名づけられたフレームワークを使用して、OSAを動的にロード、アンロードすることができる。OSAレジストリフレームワークはAdapterNamespacesをロード、アンロードするための方法を提供すると共に、システムのその他の部分に、どのオブジェクトシステムが支援されるかを確定するためのオブジェクトシステムとは無関係の方式を提供する。OSAはシステムの実行中にロード、アンロードされることができ、支援されるオブジェクトシステムの範囲を動的に拡張する。

本発明のその他の実施例

その他の実施例の範囲にある発明。別の実施例においては、たとえば、次のようなシステム及び方法が提供される。

(A) 「汎用オブジェクト (universal object)」、すなわち、複数のオブジェクトシステムで同時に実行されるように見え、且つどのオブジェクトシステムを支援するかを動的に変更することができる単一のオブジェクトの生成を可能にするシステム及び方法である。このようなシステムは「汎用オブジェクト」の生成を個別構成、アプリケーション構成及びサーバ構成で可能にする。更に、そのようなオブジェクトは、解釈済言語技法又はコンパイル済言語技法に基づいていれば良い。

異種オブジェクトシステムがインタオペレートすることを可能にするのに加えて、このようなシステムは、わずかに変形した構成で使用された場合、オブジェ

クトがいくつかのオブジェクトシステムで同時にインプリメントされるように見せることができる。図11は、そのような構成で使用されているシステムの一例を示す。図中、独立したCOMオブジェクト69は、DSOMベースアプリケーション71においてDSOMオブジェクト70として現われているばかりでなく、ORBIXベースアプリケーション73においてもORBIXオブジェクト72として現われている。

代表的なオブジェクトシステムがそのオブジェクトを使用できるようにする構成には、「インプロセス(in process)」と「アウトオブプロセス(out of process)」の2つがある。たとえば、OLE2.0 Programmers Reference及びIBM SOMobjects Developer Toolkit Programmers Reference Manualを参照。オブジェクトを「インプロセス」で使用する場合、オブジェクトは共用ライブラリと呼ばれることも多い「ダイナミックリンクライブラリ(DLL)」として使用され、それらのオブジェクトを使用しているソフトウェアと同じプロセススペースにおいて実行する。オブジェクトを「アウトオブプロセス」で使用する場合には、オブジェクトは、それらを使用しているソフトウェアのプロセススペースの中ではなく、オブジェクトが入っている別のアプリケーション又はサーバのプロセススペースにおいて実行する。

図11は全てのオブジェクトが別個のプロセスで実行している場合を示すが、インプロセスのオブジェクトと、アウトオブプロセスのオブジェクトの双方を支援するオブジェクトシステムについては、システムをいずれの構成で使用しても良いことは当業者には自明であろう。すなわち、システムがアプリケーションにより「インプロセス」で使用される場合、様々なフレームワークと、アプリケーションによって使用されているオブジェクトシステムのOSAと、アプリケーションによって使用されているプロキシオブジェクトとを含めたシステムはDLLとしてアプリケーションのプロセススペースの中で実行する。尚、プロキシは「インプロセス」であるが、ネイティブオブジェクト（すなわち、プロキシに対応する実オブジェクト）は「インプロセス」でなくとも良い。それらは、ネイティブオブジェクトシステムにおいて生成された方式に従って、インプロセス又は別プロセスのいずれかになるであろう。

逆に、オブジェクトを使用しているアプリケーションがオブジェクトを「アウトオブプロセス」で生成しうる場合には、システムが別プロセスで実行しているとしても、効率の観点から、ネイティブオブジェクトを「インプロセス」で生成するであろう。この場合、インプロセスとはオブジェクトがそれらのオブジェクトを使用しているアプリケーションのプロセスではなく、システムのプロセスで生成されることを表わしている。

図12aは、汎用オブジェクトを個別構成でイネーブルするシステムの一例を示し、図12b及び図12cは、それぞれ、アプリケーション構成と、サーバ構成を示す。図12aにおいて、74は3つのオブジェクトを含むCOM DLLである。オブジェクト75は、システムを介して複数のオブジェクトシステムから同時に出たように見える単独のオブジェクト（すなわち、アプリケーション又はサーバの中にはないオブジェクト）である。対応するOSAを追加することにより、追加のオブジェクトシステムを簡単に支援できる。DLL75が1つ又は複数の個別オブジェクトを含むことは当業者には自明であろう。

図12bは、アプリケーション構成で使用されているシステムの一例を示す。システムは、複数のアプリケーションオブジェクトを含むアプリケーション76に連係されている。この例では、アプリケーションはそのオブジェクトをシステムに（OSAと同じように）直接に記述しているので、OSAは不要である。アプリケーションのオブジェクトは、システムを介して、複数のオブジェクトシステムから同時に出たように見えるであろう。あるいは、適切なOSAを提供することにより、異なるプラットフォームで異なるオブジェクトシステムを使用して同一のオブジェクトが実行されるように見せても良い。アプリケーションがオブジェクトシステムを利用するならば、OSAも利用されうるであろうということは当業者には自明であろう。

図12cは、サーバ構成で使用されているシステムの一例を示す。サーバ構成の特徴は、オブジェクトサーバ、すなわち、使用されているオブジェクトを含む1つ又は複数のプロセスがそれらのオブジェクトを使用しているアプリケーションとは別個のプロセスにあることである。図12cでは、サーバ78の中のオブジェクト77は、システムを介して、複数のオブジェクトシステムから同時に出

たように見えるであろう。

図11から図12cにおいては、オブジェクトをコンパイル済オブジェクトとして示したが、そうである必要はない。オブジェクトは解釈済言語で書き込まれたオブジェクトであっても良く、完全に解釈されているか、又は中間表現を介して実行されても良い。必要なのは、インタプリタがオブジェクトをアクセスし、操作しうる何らかの基本メカニズムを提供し、OSAが書き込まれるか、又はインタプリタがオブジェクトをシステムに直接に記述することだけである。このような方式によって得られるユーティリティは、オブジェクトをインタプリタの言語で書き込むことができ、それらのオブジェクトがシステムを介して対応するOSAを有する何らかのオブジェクトシステムから出たように見えることであり、従って、対話型言語環境の急速開発という特徴を利用できる。

別の実施例においては、(B)前記の「汎用オブジェクト」を含めて、1つ又は複数のオブジェクトシステムからのオブジェクトクラスの構築を可能にし、それらのオブジェクトクラスを生成するときに、複数のオブジェクトシステムからのオブジェクトのサブクラス化、取り込み、又はオブジェクトのインスタンスの組込みが可能であるようなライブラリ利用又は対話型の装置が提供される。

従来、オブジェクトからのアプリケーションの構成を可能にし且つ／又はオブジェクトの構成を可能にしたライブラリベース又は対話型のソフトウェアは、そのようなオブジェクトがツールによって支援される単一のオブジェクトモデルに従うことを要求していた。たとえば、Microsoft Visual Basic V3.0 Professional Features Book 1, Control Development Guide (1993年)を参照。場合によっては、1つのオブジェクトシステムからのオブジェクトを別のオブジェクトシステムの中で使用できるようにするダイナミックコンバータが開発されたが、それら全てに制約が伴っていたうえに、外部オブジェクトの構成を可能にするものはなかった。たとえば、Borland C++ V4.0のUser's Guide (1993年)を参照。

この装置の場合、システムはアプリケーション構成で使用され、アプリケーションソフトウェアは、オブジェクトからアプリケーションを構成するため又はオブジェクトクラスを構築するためのコードとして当業者には良く知られている、

以下では構築ソフトウェアと呼ばれるコードである。アプリケーション又はオブジェクトクラスを構築するときには、構築ソフトウェアはいずれかのオブジェクトシステムからのオブジェクトクラスをO S Aと共に（構築ソフトウェアの独自の内部オブジェクトモデルに従うO S Aがあれば、そのO S Aに加えて）ロードする。構築ソフトウェアは外部オブジェクトクラスをシステムフレームワークを使用して直接に操作するか、或はそのオブジェクトモデルからのメカニズムを使用して操作する。システムにより提供される複数のオブジェクトシステムからのオブジェクトを操作する手段が一様であるため、装置がイネーブルされる。

別の実施例においては、（C）アプリケーション及びサーバの実行中に、オブジェクトクラス及びオブジェクトを利用しているソフトウェアも実行中である場合でも、そのソフトウェアを崩壊させずに、1つ又は複数のオブジェクトシステムの中でアプリケーション及びサーバの間でそれらのオブジェクトクラス及びオブジェクトの場所を再指定することを可能にするシステム及び方法が提供されている。

「インプロセス」オブジェクトと共に使用する場合、あるいは個別構成、アプリケーション構成又はサーバ構成で解釈言語環境と共に使用する場合に、1つ又は複数のオブジェクトシステムにおけるアプリケーション間及びサーバ間でのオブジェクトクラス及びオブジェクトの場所の再指定はイネーブルされる。オブジェクトクラスを移動させるときには、まず、新たな場所／オブジェクトシステムにおいてそのクラスの1つのバージョンが利用可能でなければならない。これは、ネイティブイオブジェクト定義を移動させること。インプロセスオブジェクトの場合、それらが入っているDLLをコピーするか、又は他の手段によって移動させなければならない。解釈環境の場合には、ソース又は中間表現をコピーするか又は他の手段によって新たな場所へ移動させ、適切なアプリケーション又はサーバにロードしなければならない。

システムを使用して、新たな場所及び／又はオブジェクトシステムのクラスを露出させること。

新たな場所及び新たであると思われるオブジェクトシステムを反映させるために、V C l a s s D a t a及びその他の記述要素を含めて、システム中のオブジ

(46)

ェクトクラスの記述を更新すること。

クラスのインスタンスが既に存在しており、移動させるべきでない場合には（すなわち、新たなインスタンスのみを新たな場所／オブジェクトシステムに位置させるべき場合）、そのクラスのバージョン番号を増分しなければならない。によって実行される。

新たなインスタンスが生成、操作されるとき、それらは新たな記述を利用して生成、操作されるので、新たな場所及びおそらくは新たなオブジェクトシステムについて生成及び操作は適正に行われる。ネイティブオブジェクトに関して、システムがオブジェクトの生成及び操作を実行するために利用されるのであればこの方法は外部オブジェクトと、ネイティブオブジェクトの双方に適用される。

オブジェクトインスタンスも同様のメカニズムを使用して移動される。インスタンスに関わる既存のあらゆるプロキシ（すなわち、オブジェクトシステムごと及びユーザごとのプロキシ）は、インスタンスの新たな場所及び／又はオブジェクトシステムを反映するために、ネイティブオブジェクトのソフトウェア記述を単純に更新させなければならない。

ここでは実例を示すために本発明の特定の実施例を説明したが、本発明の趣旨から逸脱せずに様々な変形を実施しうることは理解されるであろう。従って、以下の発明の範囲は添付の請求の範囲及びその等価物によってのみ限定される。

ビジュアルエッジ・ソフトウェア(Visual Edge Software)社刊

## Class Registry

## 機能仕様書

概要 (Overview) .....	42
クラス階層 (Class Hierarchy) .....	44
この仕様書がどのように構成されているか .....	50
タイプ及び一覧(Type and Enumerations).....	52
サポートクラス (Support Classes) .....	58
Class VcrHelp .....	58
Class VcrCallException .....	60
Class VcrDataRef .....	61
基本クラス (Base Classes) .....	65
Class VcrBase .....	66
Class VcrToplevel .....	71
露出されたクラス (Exposed Classes) .....	73
Class VcrArgument .....	73
Class VFunctionData .....	74
Class VPropData .....	78
Class VInstanceData .....	80
Class VExceptionData .....	81
Class VClassData .....	82
Class VAdapterNameSpace .....	92
Class VViewNameSpace .....	101
Class VClassRegistry .....	117
実行クラス (Implementation Classes) .....	119
Class VcrCodedFunction .....	119
Class VcrCodedProp .....	122
Class VcrCodedClass .....	123
Class VcrSimpleAdapter .....	132
Class VcrCodeAdapter .....	133
Class VcrSimpleView .....	134
Class VcrFlatView .....	136
管理型クラス (Type Management Classes) .....	136
Predefined Types .....	136
Class VTypeData .....	138
Class VcrFundamental .....	144

Class VcrAlias	145
Class VcrPointer	145
Class VcrObjectRef	146
Class VcrFunctionPtr	147
Class VcrStructItem	147
Class VcrStruct	148
Class VcrEnumItem	149
Class VcrEnum	150
Class VcrSequence	151
Class VcrArrav	153
Class VcrStructSequence	154
Class VcrString	155
Class VcrAny	155
Class VcrUnionItem	157
Class VcrUnion	158
Class VcrStructUnion	158
Class VcrOpaque	159
Class VTypeManager	162
ミキシン (Mixins) .....	167
Class VcrQueryMixin	168
Class VcrExposureMixin	169

## 概説

透過 (transparent) オブジェクトシステムサポート

「今日出現しつつある大域経済に参入するには. . .」 Business Week

多重プラットフォーム上を移動する製品における種々のオブジェクトシステムをサポートすることは伝統的に問題が多かった。同一オブジェクトシステムが全てのプラットフォーム上において常に利用可能であるとは限らない。1つのプラットフォームにおいて選択されたオブジェクトモデルが他のプラットフォームにおいても共通して選定されるとはとは限らない。この多様性は現存し、そして、予測可能な将来においても継続するものと仮定すれば、アプリケーション・デベロッパにとっては、多重 (multiple) オブジェクトシステムをサポートすることが当面する問題である。

多重オブジェクトモデルをサポートすることは、それ自体の問題を導入する。例えば、OLE及びDSOMのような、オブジェクトシステム用のコードを書くことは一般に複雑かつ時間のかかる作業である。更に、各個別オブジェクトシス

テムの複雑な事情を学習するために必要な投資はかなりの額に達する。更に、オブジェクトモデルに対するコード化は、もとのコードの広範囲に亙る構成し直しに関係しかねない。決定的な「厄介事」は、或るアプリケーションを或る1つのオブジェクトモデルに対してコード化することが、同時に、別のオブジェクトモデルに対するコード化の妨げとなることである。別のオブジェクトシステムに対してコード化し直すことは、ここでも、多額の投資を必要とする。

「得策がある．．．」 R o n C o I n d u s t r i e s P L C

V i s u a l E d g e クラスレジストリは、これらの問題を排除し、そして、多重オブジェクトモデルをサポートするためにデベロッパが強要される投資額を大幅に軽減する。クラスレジストリは、クラス、方法、特性、及び、データタイプを記述するために、共通APIの背後の各オブジェクトシステムを抽象する。APIは、ユーザー既存のクラス階層に容易にインタフェースするように設計される。ユーザーのクラス階層がクラスレジストリに一旦記入されると、何等修正の必要なしに、新規なオブジェクトシステムのサポートが行われる。個別オブジェクトシステムのサポートはランタイムにおいてインストール及び除去さえ可能である。

クラスレジストリは、オブジェクトシステムアダプタを介してオブジェクトシステムにインタフェースする。オブジェクトシステムアダプタは、当該アダプタがサポートするオブジェクトシステムの詳細を管理する責任がある。このオブジェクトシステムの詳細は、アダプタを越えて露出(expose)される必要は一切無い。一旦、オブジェクトシステムアダプタがクラスレジストリにインストールされると、当該オブジェクトにとって既知であるあらゆるクラスは、必要に応じて、クラスレジストリ内に透過的に導入可能であり、また、その逆も可能である。オブジェクトシステムアダプタは、クラスレジストリ内に記入された任意のオブジェクト例を当該アダプタが表すオブジェクトシステムに対して露出する全

ての詳細を処理する。更に、アダプタは、そのオブジェクトシステムから任意のオブジェクト例をクラスレジストリに戻す露出の詳細を管理する。

「しかし待て．．．更に．．．」 G i n s u K n i v e s I n c .

クラスレジストリユーザが任意のオブジェクトシステムからオブジェクトを透過的に操作することを可能にする同じファンクション性が、当該ユーザがオブジェクトシステム「ブリッジ」として作用することを可能にする。クラスレジストリは、1つのオブジェクトシステムから他のオブジェクトシステムへの通路を提供する。これは、例えば、クラスレジストリをブリッジとして使用することにより、OLE自動化アプリケーションがDSOMを制御すること、及び、その逆を可能にする。或いは、他の例として、このファンクション性は、両端における分散計算およびクラスレジストリをOLE〈一〉DSOMブリッジとして使用することにより、OLE自動化アプリケーションが、「別の」マシンのOLE自動化オブジェクトを制御することを可能にするはずである。

オブジェクトシステムアダプタを介してクラスレジスタによってサポートされたオブジェクトシステムは、「オブジェクトシステム」の従来の定義に制限されない。オブジェクトアダプタは、データベースをクラスレジストリにインタフェースするように作成可能である。従って、このデータベースは、任意の標準オブジェクトシステムから容易にアクセス可能である。更に、クラスレジストリは、所有権オブジェクトシステム用のオブジェクトシステムアダプタを付加することにより標準オブジェクトシステムを所有権オブジェクトシステムにインタフェースするために使用できる。他のオブジェクトシステムにおけるオブジェクトにアクセスしながら、同時に、それらのオブジェクトシステムを継続して使用することを可能にするので、これにより、所有権オブジェクトシステムのユーザに有力なレバレッジを提供することとなる。

クラスレジストリは、インストールされた全てのオブジェクトシステムを、1つの簡単な単一化された環境として、表すことが可能である。ただし、オブジェクトシステムの別のユーザは別の必要性を持つ。一方、幾らかのユーザは、全てのクラス及びタイプに関してフラットなネームスペースを欲しがり、他方、別のユーザ達は、クラス及びタイプネームの厳密な階層的グループ分けを希望することがあり得る。更に、別のユーザは、彼らの現行タスクに更に密接に関係するネームグループ分けの彼ら自身の階層を定義することを選定することもあり得る。クラスレジストリは、ビューを用いることによりこれをサポートする。

ビューは、オブジェクトシステムアダプタによって定義された実際の階層から独立したネームスペース階層を定義する。異なったタスクに対して、ユーザーが彼のクラスを、異なった方法において、組織化出来るように、アダプタに基づく階層に関して異なる多重ビューを作成することが出来る。ビューは、それらの中のクラス及びタイプを参照するコードに影響を及ぼすことなしに、組織化し直すことも可能である。最後に、デベロッパは、彼らのユーザーに対して持続性のあるネームスペース階層を提供するために、

ビューをサブクラスすることが出来る。

「私にとって一体何の意味があるのか？」 K T e l   M a r k e t i n g   L t d .

クラスレジストリは、あらゆるオブジェクトシステムにインタフェースするベースファンクション性を提供する。これは、デベロッパに対して彼らの資源を特定の強度に自由に集中することを可能にする。クラスレジストリの単一ポータブルAPIの結果として、新規なオブジェクトシステムが現れ、そして、他のシステムがすたれた場合に、アプリケーションのライフサイクルに亘って開発およびメンテナンスコストを低減させる。これは、市販するための全投資額および時間の大幅節減を意味する。

#### クラス階層

クラスレジストリは、クラス、ファンクション、及び、タイプのIDLまたはC++ヘッダファイルと同じ目的のために、これらよりも遥かに役立つ。即ち、クラスレジストリは、C++クラスを通じてクラス及びタイプ階層を表す。クラス及びタイプは、静的、動的、および、データベース様資源からクラスレジストリ内に入ることが出来る。クラス及びタイプ定義の静的資源は、一般に、手動でコード化され、そして、「コードアダプタ」に記憶されたクラスレジストリエン트리である。例えばOLE及びDSOMのような動的資源は、それらの固有レポジトリを介してクラスレジストリにクラス定義を提供する。例えばタイプライブラリのようなデータベースソースは、ロードクラスをクラスレジストリ内にデマンドするメカニズムを提供する。

クラス階層の定義と同様に、クラスレジストリは、クラスインタフェースを言語インタプリタ、ビルダ、オブジェクトブローザ、及び、特性エディタにアダプタイズする柔軟な動的方法を提供する。クラスレジストリ内のクラスは、全体としてブローズ可能であるか、又は、ネーム、バージョン番号、オブジェクトシステムアダプタ、等々別に要求可能である。クラスレジストリのユーザ（インタプリタ、ビルダ、等々）は、当該クラスのソースに接続される必要はない。クラスは、クラスレジストリ内にインストールされた任意のオブジェクトシステムから入来可能である。

クラスレジストリは、オーバーヘッド層の下の外部オブジェクトシステムにおいて定義されたオブジェクトを隠さない。クラスレジストリは、そのクラスインスタンスを、当該インスタンスへの実際のポインタによってマニピュレートする。この場合、変換またはキャストは一切不要である。これにより、クラスレジストリは、それらの固有オブジェクトシステムがオブジェクトにアクセスする速さと同じ速さによってオブジェクトにアクセスすることが可能である。

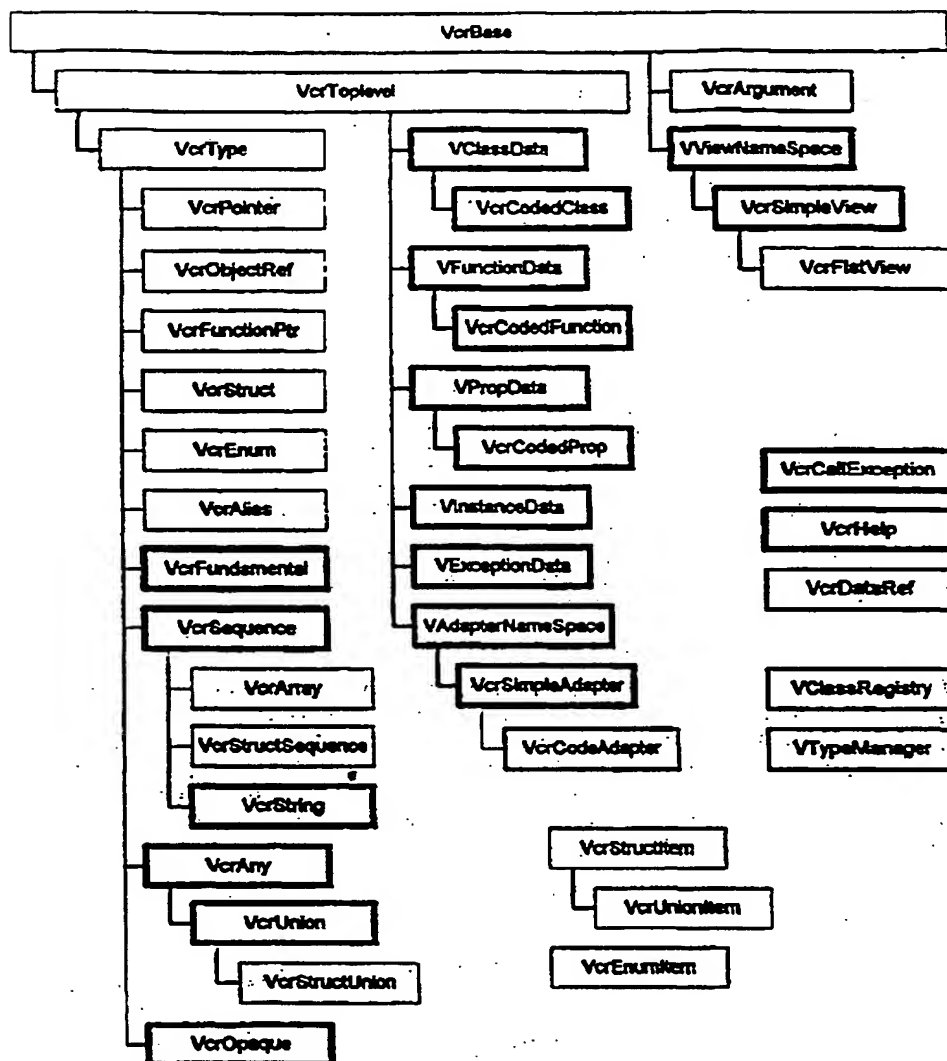
クラスレジストリにおけるクラス記述は、クラスネーム、そのベースクラス（単数または複数）、及び、その方法、ユーザーがアクセス可能な特性、ネストされたタイプ、例外、及び、名前付き定数、又は、例のリストを有する。

各方法および特性は、生きたオブジェクトのメンバーに実際にアクセスするためのフックを伴った幾らかの情報（名前、データタイプ、及び、それらのタイプのアーギュメントのリスト）を有する。各方法記述は、当該クラスレジストリを使用するソフトウェアが方法呼び出すためにコール出来るファンクションポインタを有する。各特性記述は、2つのアクセッサ方法記述を含むことが出来る。そのうちの一方は値を得るための記述であり、いま一方は値を設定するための記述である。読み取り専用または書き込み専用特性は、アクセッサ方法の一方を省略しても差し支えない。

Cコール可能ファンクションは一般的に用いられる。コンパイルされた殆どの言語は、外部ファンクションコールのためのCをサポートする。C++法およびデータメンバーへの直接アクセスを与えることは便利ではあるが、当該言語によ

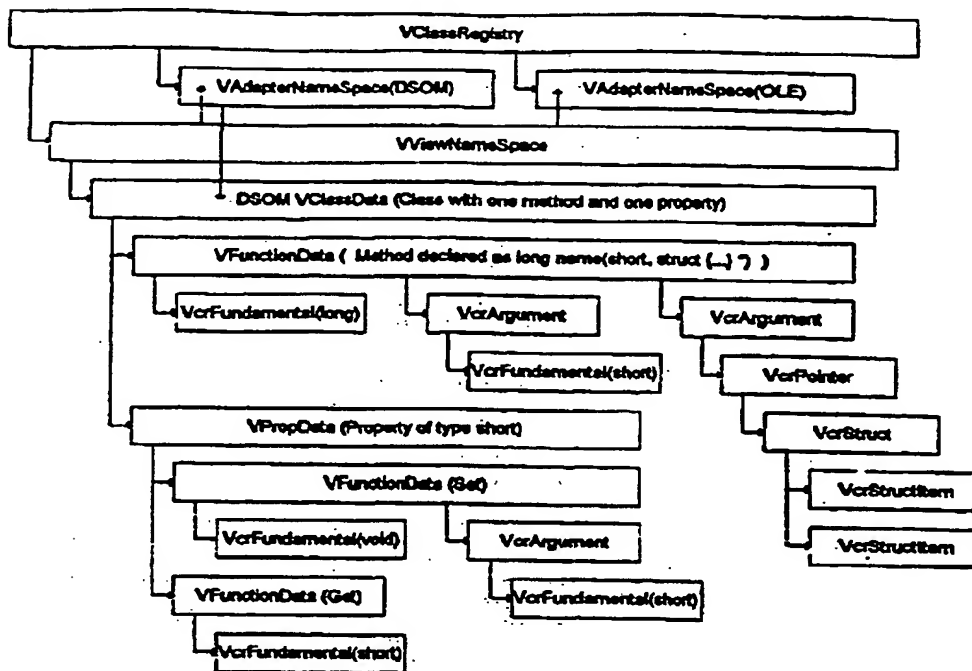
って課せられる制限条件があるために、クラスレジストリから独立して開発されたクラスにとっては実用的でなくなる。C++法の実現が属性として断定可能である（例えば、マイクロソフトオブジェクトマッピングを使用するプラットフォーム）状況の下においては、仮想的な方法の直接レジストレーションがサポートされる。これと同じ理由により、データメンバーは、直接オフセット又はメンバーポインタでなくて1対のアクセッサファンクションによって表される。

次のダイアグラムは、クラスレジストリのクラス階層を示す。太い線で囲まれたクラスは、クラスレジストリのユーザーによってサブクラスされることが可能であるべきものとして指定される。



(54)

クラスレジストリは、クラスを定義するために広範囲に亙る例階層を提供する。この階層の理解を容易にするために、見本としてのインスタンス階層を次の図に示す。この図は、クラスレジストリ及びその構成部分のインスタンスレイアウトの見本を示すに過ぎない。この図は、完全であることを意図するものでなく、インスタンスレイアウトの概観を示すに過ぎない。実線矢印はオブジェクトへの直接ポインタを表す。この階層は、インストールされた2つのオブジェクトシステムアダプタを持つクラスレジスタを表す。クラスレジスタは、1つの方法と1つの特性を持つ1つのクラスを有する。



### 静的エン트리 (Static Entries)

クラスレジストリは、方法および特性、タイプ、ファンクション、例外、及び、名前付きインスタンスを伴った静的に定義されるクラスの作成をサポートする。静的インスタンスは、クラス (`VcrCodedClass`、`VcrCodedFunction`、`VcrCodedProp`、`VInstanceData`、`VExceptionData`のインスタンス) を例にとることによって定義される。これらのクラスは、インストールの方法およびクラスにおける特性をセットする、アーギュメントを加える、等々のためにAPTを提供する。クラスは、

VCodeClassオブジェクトを作成することにより定義される。方法および特性（VCodeFunction及びVCodePropインスタンス）は、方法（AddMethodおよびAddProperty）を用いて当該クラスに加えられる。

これらの方法は、AddArgumentを用いて当該方法に加えられる1組のアギュメント（VcrArgumentのインスタンス）によって定義される。方法を定義する場合には、Cコール可能なファンクションが提供されなければならない。

クラス方法用ラッパーファンクションを使用すると、特にスクリプトに際して、プログラム可能なオブジェクトの統合を一層容易にすることが出来る。ラッパーアギュメントは、下位クラスのアギュメント用に用いられたタイプと厳密に同じである必要はなく、ラッパーは、より簡単なタイプ（例えば「int.」のような）から、例えばBasicのような言語において表現することが困難な列挙法のような特殊なタイプに変換することが出来る。読み取り専用特性は、NULLセットファンクションを供給するか、又は、特性使用フラグを「読み取り専用」にセットすることによりレジスタすることが出来る。

各Cラッパーファンクションのタイプは、当該タイプがその第1アギュメント（明示の「これ」）としてのインスタンスポインタを伴ったファイルスコープファンクション（又は、静的メンバー）である場合を除き、当該タイプが表すメンバーファンクションにマッチする。マクロは、C++クラスメンバー用ラッパー及びクラスレジストリアイテム構成コードの両方を生成する物として定義できる。

#### 動的エントリ

静的エントリの取り扱いに加えて、クラスレジストリは、高度に動的なエントリを取り扱うことが出来る。例えば、対話的オブジェクト開発またはオブジェクトシステムブリッジングにおいて、ユーザーが外部クラスに関してブローズおよび要求する場合、又は、ユーザーが自身のクラスを加えるか、又は、変更する場合に、クラスレジストリの内容は変化可能である。クラスレジストリが、動的で

あるオブジェクトシステムに接続されている場合には、クラスレジストリ自体が動的である。

例えば言語のインタプリタのような開発環境は、コンパイルタイム、ランタイム、又は、両者における全ての外部タイプ及び名前結合情報に関して、クラスレジストリに相談することができる。パースタイムにおける質問の結果は、単に、回答がランタイムまで延期されることを指示するに過ぎないこともあり得る。クラスが静的に定義される場合には、これらクラスの定義は、コンパイルタイムにおいて入手可能である。例えばDSOM又はOLEのようなオブジェクトシステムからクラスがインポートされる場合には、これらのクラスはコンパイルタイムにおいて入手可能であるが、全てのこれらの方法および特性は、クラスがアクセスされる時まで入手出来ないことも有り得る。更に、クラスは、データベースアダプタから検索することも出来る。当該クラスがローカルキャッシュ内において見付からない場合には、データベースアダプタは、クラスの定義を見付けるために、データベースへの質問を行うことも出来る。

クラスレジストリの質問インタフェースは、抽象クラスVViewNamespaceによって定義される。名前結合サービスは、その仮想的方法を介して提供される。一方において、クライアントプロセスは、一般に、1つのVCClassRegistryインスタンスによってされ、他方において、サーバープロセスは、クライアント1人当たり1つのクラスレジストリインスタンスを持つことが出来る。一般に、1つのプロセスによって維持されるクラスレジストリは、特殊化された1組のアダプタを管理する。この場合、各アダプタは、クラス及びタイプ情報の異なるソースに質問することができる。内部クラスレジストリデータは、これら他のソース（例えばインタフェースレポジトリ）から出て来る情報に関して、僅かにキャッシュ以上のサービスを提供する。

各オブジェクトシステムの特種必要条件には、クラスレジストリアイテムクラスのサブクラスを定義することによって適合可能である。レジストリアイテムクラスは、クラスレジストリの構造およびそれを検査するためのAPIを定義する。レジストリは、必ずアイテムクラスの仮想メンバーファンクションを介して相談を受ける抽象APIである。

### `mix in` (ミキシン) 拡張性 (extensibility)

クラスレジストリは、当該クラスレジストリ自体のクラスに関する `mix in` フレームワークを定義する。クラスレジストリ内の各アイテムは、`mix in` によって拡張可能である。クラスレジストリエントリに関する `mix in` は、或るオブジェクトシステムにおいては利用可能であるが、全てのオブジェクトシステムによってサポートされるとは限らないファンクション性に対するサポートを提供する。更に、`mix in` は、クラスレジストリの基本的実現性を越えてそのファンクション性を強化するためにも使用することが出来る。例えば、クラスレジストリからそれらのオブジェクトシステムへのクラス露出は、全てのオブジェクトシステムによってサポート可能というわけではない。従って、露出をサポートするオブジェクトシステムは露出 `mix in` を提供することが可能であり、他方、露出をサポートしないオブジェクトシステムは露出 `mix in` の実現を提供しない。`mix in` メカニズムは、ベースシステムモジュールをコンパイルし直すことなしにクラスレジストリを完全に拡張可能にする。

`Visual Edge` は、`mix in` の規格適合性 (Apple Event Suite に類似的) を定義するために、そのパートナーと共に作用する。これにより、アプリケーションのインターオペラビリティを強化し、アドオンツール用の強力市場の促進を支援する。

`mix in` は、`mix in` が必要でない場合には DLL をロードする必要がないように、外部 DLL において提供可能である。これは、更に、システムがそれ自体の「アドオンパッケージ」を定義することを可能にする。アドオンパッケージは、`mix in` の実現を含む DLL であっても差し支えない。DLL が見付からない場合には、`mix in` は提供されず、システムは、そのデフォルト動作を継続する。

更に、`mix in` 拡張性メカニズムは、メモリ消費の節約も提供することが出来る。`mix in` は、最初に要請された場合に、動的に作成される。`mix in` は、必要である場合に限って作成される。

クラスレジストリは、アプリケーション及びその中において `mix in` が実現されるオブジェクトシステムとは独立した `mix in` フレームワークを提供する

能力を提供する。即ち、アプリは、それ自体のオブジェクトシステムにおける抽象クラス及びmixinのフレームワークを定義する。アプリが「クラスレジストリアウェア」である場合には、第三者は、彼らが選定したオブジェクトシステムにおけるフレームワーククラス又はmixinの彼らによる実現を開発することが可能である。ビルダーは、例えばフォント及びカラーのようなオブジェクトを特殊化して編集するためにmixinを定義することが出来る。本規格のmixinの定義を採用したとすれば、第三者は、ビルダーがDSOM上においてさどうする場合であっても、規格mixin定義を用いて、OLEにおいて特殊化されたエディタの実現が可能である。これにより、ユーザーは、自由に選択が可能であり、ユーザーのアプリのインターオペラビリティが増大する。

#### 本仕様がいかに構成されているか

この仕様の残りの部分は、クラスレジストリによって用いられるクラス及びタイプを定義する。

- \* 第1セクションは、クラスレジストリ全体を通じて使用される列挙表およびtypedefsについて記述する。これらは、複数の場所において、そして、複数の目的のために使用される一般的なタイプである。個別クラスに特有のtypedefsは、クラス自体と共にリスト表示される。
- \* 第2セグションは、ヘルプサポート、データリファレンス、及び、ファンクションコールエラーリターンのための一般的クラスを定義する。これらのクラスは、例えば、記述およびヘルプサポート、エラー報告、及び、データリファレンスカウンティングのような、特殊ヘルパーファンクション用として用いられる。
- \* 第3セクションは、クラスレジストリのクラス用の基礎として用いられるベースクラスについて概説する。これらのクラスは、例えば、各要素に関する使用コード及びヘルプ情報に関する普及情報のためにPAIサポートを提供する。
- \* 第4セクションは、クラスレジストリのユーザーに露出されるクラスを定義する。これらのクラスは外界に対するAPIであるが、実現は実際にサブクラスしても差し支えない。
- \* 第5セクションは、露出された抽象クラスの実現であるクラスについて記述

する。これらのクラスは、直接例挙またはサブクラス化によって使用するために使用可能である。

\* 第6セクションは、クラスレジストリ用タイプ管理階層を定義する。これらのクラスは、タイプのキャスト及びオブジェクトシステム間の比較のためにサポートを提供する。

\* 最後のセクションは、現在定義済みのエクステンションについて概説する。これらのエクステンションは、ファンクション性をサポートする特定のクラスレジストリエントリにおける `mix in` として利用可能である。

クラスレジストリは、次に示す `Visual Edge` ユーティリティクラスを使用する。これらのクラスは、本ドキュメントにおいては定義されない。クラスレジストリを使用するソフトウェアはこれらのクラスを定義するライブラリへのアクセスを必要とする。

`VArray` 拡張可能なタイプされたアレイ。

`VAtomDictionary` 原子をキーとするハッシュテーブル。

`VClass` ランタイムタイプの識別用サポート。

`VValue` カプセル入りユニオンオーバー整数タイプ。

`VString` カプセル入りヌル終結Cストリング。

`VAtomRef` 高速ハッシング及び比較のために使用されるストリング  
アイデンティファイア。

`VPrimary mix in` サポート用ベースクラス。

`VrPrimary` 基準カウンティング及び `mix in` 用ベースクラス。

`VMix in mix in` 実行用ベースクラス。

`VrReference` 当該オブジェクトの基準カウントを追跡するオブジェクトに対するスマートポインタテンプレート。これは、適切な多重スレッドサポート用として必要である。

`VrRefList` 当該オブジェクトの基準カウントを追跡するオブジェクトに対するポインタのアレイのテンプレート。これは、適切な多重スレッドサポート用として必要である。これ

は、V e R e f e r e n c e オブジェクトのアレイではないことに注意されたい。

### Type、及び、Enumeration (列挙)

#### 使用コード

レジストリは、殆どのアイテムにコードを関連付ける。使用コードはV T U s a g e C o d e タイプであり、そして、例えば、インタプリタは操作できるが、ユーザーにとって直接アクセス可能にしてはならない特性に関する「隠し」のようなアイテムをどのようにして使用できるかについて記述するビットフラグの組み合わせである。

使用コードは注釈および具体化の詳細を供給する。例えば、リファレンスによるコール対値によるコールアーギュメントを指定するためには、個別タイプは使用しない。使用ビットは、リファレンスによるコールとしてファンクションアーギュメントをマークする。

使用ビットは、文脈特有であることをフラグし、そして、クラスの異なるアイテムに関して異なる意味を付与することがあり得る（同じビットが、特性に関しては一方の物を意味し、クラスに関してはもう一方の物を意味することもあり得る）。どの使用コードがどのクラスに供給し、そして、更に、どのユーザーが当該使用コードを最も参照しそうであることを要約して次の表に示す。

(61)

使用するコード	C	F	M	P	E	T	I	N	A	L	B	V
KVUsageReadOnly				.			.			.	.	
KVUsageWriteOnly				.			.			.		
KVUsageDefaultProperty				.						.		
KVUsageProtected			.	.	.	.	.			.		
KVUsageThreadSafe	.	.	.	.						.		
KVUsageStrict		.	.	.					.	.		
KVUsageVarArg		.	.							.		
KVUsageCallerIsOwner		.	.						.	.		
KVUsageByRef		.	.						.	.		
KVUsageHidden		.	.						.	.	.	
KVUsageOptional									.	.		
KVUsageHasDefault									.	.		
KVUsageValue									.	.		
KVUsageOut									.	.		
KVUsageInOut									.	.		
KVUsageDynamic	.									.		
KVUsageDeleteNotify	.											.
KVUsageCannotSubclass	.									.		
KVUsageDoNotCache	.	.			.	.	.				.	.

## Legend

C	VClassData	F	VFunctionData
M	VFunctionData(method)	P	VPropData
E	VExceptionData	T	VTypeData
I	VInstanceData	N	VAdapterName Space
A	VcrArgument	L	Used by a language
B	Used by a property browser	V	Used by a view or adapter

使用するコードの意味は以下の通りである。

KVUsageReadOnly 特性がユーザーがアクセス可能なセット方法を持たない。

KVUsageWriteOnly 特性がユーザーがアクセス可能なゲット方法を持たない。

KVUsageDefaultProperty オブジェクト自体が値式に現れた場合に、オブジェクトの1つの特性を、割り当てられるか、或いは、読取られるべき特性（アサインメン

(62)

トの受取人として) として区別する。

**KVUsageProtected** クラスのこのエレメントが保護されたメンバーであることを指定する。

**KVUsageThreadSafe** この使用フラグは、このファンクションまたは方法が同時に多重スレッドによってコールされ得ることを指定する。

**KVUsageStrict** 正常な言語ルールによって強制されてはならない特性、アーギュメント (引数)、ファンクションリターン、または、方法戻し。

**KVUsageVarArg** ファンクションまたは方法に関して、一切のアーギュメント記述がそのために利用可能ではないことを示す。ファンクションまたは方法は、スクリプトされたソースにおいて発見された際に、コールアーギュメントを用いてコールされなければならない。可変個数のアーギュメントを持つものとして記述されたファンクションまたは方法は他のオブジェクトシステムに露出可能であってはならない事に注意されたい。

**KVUsageCallerIsOwner** この使用フラグは、このファンクションリターン、方法リターン、アーギュメントにおいて戻される値がコーラーにより解放されるべきであることを指定する。

**KVUsageByRef** アドレスによって実際にパスされるべきアーギュメントを識別する。

**KVUsageHidden** ファンクション、方法、アーギュメント、または、特性は、ユーザーでなくて環境によって供給される。

**KVUsageOptional** ファンクション、または、方法アーギュメントは、コールにおいて省略され得る。

`KVUsageHasDefault` コールから省略された場合には、`VcrArgument`からのそのためのデフォルト値がコールにおいてパスされなければならないアーギュメント。

`KVUsageValue` 特性に関して実際の値が保持されるべきアーギュメントまたは特性の集合に関して指定される。特性の集合が多重アーギュメントをとる場合に限りこれが必要とされる。

`KVUsageOut` アーギュメントは、実際にファンクションまたは方法からのリターンである。

`KVUsageInOut` アーギュメントは、ファンクションまたは方法への入力であるが、コール期間中に修正されても差し支えない。

`KVUsageDynamic` この使用は、このクラス定義が質問の間で変化するかもしれないことを指定する。即ち、クラスが質問される場合には、当該クラスは、最初に質問された場合よりもより多くのエレメントを持つこともあり得る（特性、方法等）。エレメントは、取り去られることなく加えられることだけが可能である。

`KVUsageDeleteNotify` これは、このプロセススペースにおいてこのオブジェクトのインスタンスが破壊されるか、或いは、十分にデリファレンスされるた場合には、もう一方のアダプタがこれについて通告されることを指定する。

`KVUsageCannotSubclass` このクラスは、言語またはビルダによってサブクラスされることは不可能である（ただし、その生来のオブジェクトシステムにおいてはサブクラスされることもあり得る）。

`KVUsageDoNotCache` この使用フラグは、このトップレベルア

アイテムが任意のビューによって隠されてはならないことを指定する。

割り当てられていない利用可能な使用コードの集合はかなり小さい。使用コードは、他の定義と重複しないように、必要に応じて、Visual Edgeによって割り当てられなければならない。

`Enum VTCallType`

この列挙は、方法またはファンクションを正しく呼ぶために要求される呼出しコンベンションを定義するために使われる。あらゆる所定のアーキテクチャに関して、後続する呼出しコンベンションの幾つか、又は、全てが重複し、そして、同じ具体化を提供しても差

し支えない。

`KVCallTypeC` C呼出しコンベンションによりファンクションをコールすること。この呼出しコンベンションは、プロトタイプANSI Cなしで宣言されるか、或いは、可変個数のアーギュメントと共に宣言されたファンクション用に用いられなければならない。

`KVCallTypeAnsic` ANSI C、または、C++呼び出しコンベンションによりファンクションををコールすること。この呼出しコンベンションは、ANSI Cプロトタイプ及び固定数のアーギュメントにより宣言されたファンクション用に使用されなければならない。

`KVCallTypePascal` パスカル呼び出しコンベンションによりファンクションをコールすること。この呼出しコンベンションは、パスカルによって書かれるか、或いは、パスカル呼び出しコンベンション（即ち、`__pascal`）によって宣言されたファンクション用に使用されなければならない。

`KVCallTypeInternal` ANSI C呼び出しコンベンション

(65)

ン、及び、内部のコールサインによってファンクションをコールすること。内部のコールサインを次に示す：

```
VcrCallException *InternalCallsignature(
    VFunctionData      *funcData,
    unsigned            paramCount,
    VTypeData          **paramTypes,
    void                **params,
    VTypeData          *resultType,
    VcrDataRef          *result,
```

`funcData` コールのためのファンクション記述。

`paramCount` パラメータの個数。

`paramTypes` パラメータのタイプ。

`params` パラメータに対するポインタの配列。

`resultType` 要求されたリターンタイプ。

`result` (戻し) ファンクションのリターン値に対するポインタ。

全てのアーギュメントに関する値が指定されなければならない。隠されたアーギュメントも含まれなければならない。アーギュメントのうちの1つでも期待したタイプに変換出来ない場合、或いは、ファンクションの実際のリターンタイプが所要のリターンタイプに変換出来ない場合には、当該ファンクションは、フェールし、そして、例外を返さねばな

らない。

`Enum VTSearchType`

この列挙は、クラスレジストリエントリを探索する方法を指定するために用いられる。オブジェクトシステムアダプタは、クラスまたはファンクションに関しておそらく長く複雑な探索を行うべきかどうか、或いは、クラスまたはファンクションに関して簡単に局所的に見るべきであるかどうかを決定するために、このアーギュメントを使用しなければならない。タイプとファンクションの遅延したバインディングをサポートする言語は、現在、定義が存在するかどうかを知るために、初期においては、局所的な探索を実施しても差し支えない。クラスまた

(66)

はファンクションが、パースタイムにおいて、見付からなかった場合には、前記の言語は、ランタイムにおいて、完全な探索を実施する。

**KVSearchAll** アイテムを見付けるために、可能な全ての方法により探索すること。

**KVSearchLocal** アイテムに関して、局所的探索のみを実施すること。アイテムを見付けるために、時間を無駄にする一切の操作を行わないこと。

**Enum VTSearchCase**

この列挙は、クラスレジストリエントリーに関する名前の探索を実施する方法を指定するために使用される。クラスレジストリエントリーのオブジェクトシステムアダプタ及びサブクラスは、名前探索の両モードをサポートすることを試みなくてはならない。ケースインセンシティブな探索が不可能である場合には、全ての探索は、その代わりに、ケースセンシティブでなければならない。

**KVCaseSensitive** 名前探索がケースセンシティブである。

**KVCaseInsensitive** 名前探索がケースインセンシティブである。

**Enum VTClassTag**

この列挙は、特定のVcrBaseインスタンスが属するVcrBaseの主要なサブクラスを決定する。これは、クラスレジストリにおいて総称ルックアップから戻されたオブジェクトのタイプを区別するために使用することが出来る。

<b>KVcrArgument</b>	<b>KVcrClass</b>
<b>KVcrFunction</b>	<b>KVcrProperty</b>
<b>KVcrType</b>	<b>KVcrInstance</b>
<b>KVcrException</b>	<b>KVcrAdapterSpace</b>
<b>KVcrViewSpace</b>	<b>KVcrAll</b>

**Enum VTTypeSubclass**

この列挙は、特定のVTTypeDataインスタンスが属するVTTypeDataのサブクラスを決定する。これは、タイプツリーの横断を容易にするために使用できる。

(67)

KVtmPointer	KVtmobjectRef
KVtmFunctionPtr	KVtmStruct
KVtmUnion	KVtmEnum
KVtmArray	KVtmSequence
KVtmVariableArray	KVtmAny
KVtmOpaque	KVtmFundamental
KVtmAlias	

## Enum VTExceptionType

この列挙は、例外を呼ぶ全ての総称クラスレジストリファンクションを定義する。

KVExceptionUnknown 例外のタイプは未知である。

KVExceptionCallFailed 例外のタイプは、コールサポートコードに特有である。オブジェクトアダプタは、配列上のエラー等々が発生した場合に、このタイプの例外を返さねばならない。

KVExceptionCalleeFailed コールは完成したが、コールされたファンクションが例外を生成した。

KVExceptionNoResources コールを完成するには利用可能な資源が不十分であった。例えば、これは、メモリを使い果たすことによって生成され得る。

KVExceptionBadparam パラメータの1つは、必要とされるタイプに変換されてはならない。

KVExceptionParamCount パラメータの個数が誤っていた。

KVExceptionBadObject コールが試みられたオブジェクトインスタンスは無効であった。

KVExceptionCouldNotExpose コールにパスされたオブジェクトインスタンスが外部オブジェクトシステムからのインスタンスであったので、このインスタンスは固有オブジェクトシステムからのインスタンスに変換出

(68)

来なかった。

`Enum VTInternalRefType`

`VTInternalRefType`は、クラスレジストリエントリーに対する内部リファレンスのタイプを定義する。

`KVDirectRef` リファレンスは階層内の他のアイテムに直接向けられる。

`KVLoopRef` リファレンスは循環性を形成しても差し支えない。

`VTObjectSystem`

タイプマネージャまたはクラスレジストリエントリと関連したオブジェクトシステムアダプタを決定するために用いられる16ビットの符号なし数として定義される。このIDは、自身のオブジェクト及びクラス対他のオブジェクトシステムからのクラス及びオブジェクトに関するあらゆるハンドリング必要条件を決定するために、オブジェクトシステムアダプタによって使用される。更に、このIDは、クラスレジストリに質問する際の探索判定基準を決定するためにも使用される。`VTObjectSystem`のIDは、`Visual Edge`によって割当てられなければならない。事前定義されたオブジェクトシステムIDは`KVAnyObjectSystem`及び`KVNoObjectSystem`システムである。

`VTVersion`

`VTVersion`は、名前は同じであるが、具体化の異なるクラスのバージョン番号を指定するために用いられる32ビットの符号付き数として定義される。更に高いバージョン番号は一層新しい定義を意味する。負のバージョン番号は違法である。一定の`KVVersionNotSpecified`は、最高バージョンが発見されるべきであることを指定するために、探索中に用いられる。

`Support` (サポート) クラス

サポートクラスは、他のクラスレジストリのクラスに対してサポートを提供する。`VcrHelp`クラスは、クラスレジストリエントリ、及び、オブジェクト方法及びエラーをコールする特性に関するヘルプ情報を提供するために用いられる。`VcrCallException`は、方法または特性へのコールが失敗す

る理由に関する情報を提供する。問題のソースについて利用可能なエンドユーザが読み取り可能な情報がある場合には、`VcrCallException`のインスタンスが`VcrHelp`のインスタンスを含んでも差し支えない。必要に応じてデータオブジェクトが削除されるように、リファレンスカウントされたデータオブジェクトを容易に維持するために`VcrDataRef`が用いられる。

クラス `VcrHelp`

`VcrHelp`は、クラスレジストリヘルプインタフェースを定義する。`VcrHelp`のサブクラスは、各プラットフォームの固有ヘルプシステムを利用するため実行されることを意図したものである。

公共メンバー

`VcrHelp` `VcrHelp`オブジェクトを構成する。

`~VcrHelp` `VcrHelp`オブジェクトを破壊する。

オーバーライド可能な公共メンバー

`Description` 所有者オブジェクトの本文記述を戻す。

`PutDescription` このヘルプアイテムのために本文記述をセットすること。

`HasHelp` アイテムが、その所有者オブジェクトに関するヘルプ情報をディスプレイでこるかどうかを指示する。

`DisplayHelp` 所有者オブジェクトに関するヘルプ情報をディスプレイすること。

`VcrHelp`コンストラクタ

`VcrHelp()`

`VcrHelp(VString description)`

`description` 所有者オブジェクトの記述。

ヘルプサポートオブジェクトを作成すること。記述を、そのアーギュメントをみなす便利なコンストラクタがある。これは、デフォルト記述情報をセットする

`VcrHelp`デストラクタ

(70)

**-VcrHelp()**

ヘルプサポートオブジェクトを破壊すること。クラスレジストリまたはタイプマネージャーオブジェクトにヘルプオブジェクトが加えられる場合には、その所有者が破壊される時に、当該ヘルプオブジェクトが破壊される。

**Description**、及び、**PutDescription**

**Vstring Description()**  
**void PutDescription(VString)**

**Description**特性は、所有者オブジェクトの本文記述を記憶する。例えば、記述の動的ルックアップを遂行することは、サブクラスにおいて再実行可能である。記述は、エンドユーザーに対するディスプレイに適するか、或いは、そうでない場合には” ” でなくてはならない。

**HasHelp**

**bool\_t HasHelp()**

**HasHelp**は、このオブジェクトが固有ヘルプシステムをサポートするかどうかを戻す。リターンが真である場合には、**VcrBase::DisplayHelp**は、自

身のオブジェクトに関するヘルプを持ち出すことができなければならない。

**DisplayHelp**

**void DisplayHelp()**

**DisplayHelp**は、所有者オブジェクトに関する情報と共に固有ヘルプシステム情報を持ち出す。オブジェクトがヘルプをサポートしない場合には、**HasHelp**は偽を戻さねばならない。

クラスVcrCallException

**VcrCallException**は、失敗したコールに関する情報をファンクションまたは方法に戻すために使われる抽象クラスである。

公共メンバー (Public Members)

**PutHelp** この例外に関するヘルプシステムオブジェクトをセットする

(71)

オーバーライド可能な公共メンバー (Overridable Public Members)

**Type** 例外のタイプを戻す。

**Code** 例外に関するエラーコードを戻す。これは、例外のソースに特有である。

**SourceName** 例外のソースの名前を戻す。

**Help** この例外に関するヘルプシステムオブジェクトを戻す。

**Index** 例外の原因となるパラメータ（有る場合には）のインデックスを戻す。

**UserType** これがユーザーによって定義された例外である場合には、例外タイプを戻す。

**UserData** これがユーザーによって定義された例外である場合には、例外データを戻す。

**Help**、及び、**PutHelp**

```
VcrHelp *Help()
void PutHelp(VcrHelp *help)
```

**help** このアイテムに関するヘルプ。

**Help**特性は、この例外に関するヘルプシステムオブジェクトを記憶する。ヘルプシステムオブジェクトは、この例外に関する記述情報、並びに、固有ヘルプシステムからのヘルプをディスプレイすることが出来る。**Help**特性がNULLである場合には、この例外は一切のヘルプをサポートしない。

**Type**

```
VTEExceptionType Type()=0
```

**type**は、例外のタイプを戻す。このタイプは、言語特定エラーメッセージに翻訳可

能である。

**Code**

```
log Code()=0
```

**Code**は、ソース特定エラーコードを戻す。この値の意味がソースに関して

(72)

既知でない限り、この値は解釈不能である。

`SourceName`

`VString SourceName()=0`

`SourceName`は、例外のソースの名前を戻す。これは、アプリケーション、オブジェクトシステム等の名前であっても差し支えなく、或いは、クラスレジストリコード内においてエラーが検出された場合には、” ” であることもあり得る。

`Index`

`long Index()=0`

`Index`は、例外を引き起こしたアーギュメントのインデックスを戻す。例外タイプが特定のアーギュメントに適用されない場合、或いは、例外を引き起こすアーギュメントが未知である場合には、この値は負である。

`UserType`

`VExceptionData *UserType()=0`

`UserType`は、ユーザーによる定義済みの例外の発生したタイプを戻す。例外に関するデータは、`UserData`により検索される。

`UserData`

`const VcrDataRef &UserData()=0`

`UserData`は、ユーザーが定義した例外と関連するデータを戻す。このデータのタイプは、例外のタイプと同じである。

クラス `VcrDataRef`

`VcrDataRef`は、オブジェクトまたはデータに関するリファレンスを管理するために使用されるクラスである。当該クラスは、自己記述であり、そして、データを引用する全ての`VcrDataRef`オブジェクトの処理が終った場合、データを適切に処理する。自動削除作動に関しては、全てのリファレンスは、直接オリジナルデータからではなく、当該データに関する第1リファレンスから構成されなければならない。内部的には、リファレンスは、オブジェクトに関する所定数のリファレンスを一時的場所に記憶するの

(73)

で、全てのVcrDataRefオブジェクトがこれにアクセスすることができる。

公共メンバー

VcrDataRef VcrDataRefオブジェクトを作成する。

Address ポインタをデータに戻す

Type データのタイプに戻す。

Value 内部VValueに戻す。

PutPointer ポインタをデータにセットする。

PutConstant データの値を定数にセットする。

putcopy 別のデータリファレンスのコピーであるように、これをセットする。

PutOwnership リファレンスに関するデータの所有権をセットする。

IsContained データがリファレンスに含まれるかどうかを戻す。

WillDelete データが削除されるかどうかを戻す。

WillDeref オブジェクトがデリファレンスされるかどうかを戻す

。

VcrDataRefコンストラクタ

```
VcrDataRef( void          *dataPtr,
             VTypeData    *type,
             bool_t        doDelete=FALSE,
             bool_t        doDelef=FALSE)
VcrDataRef( VValue        data,
             VTypeData    *type=0)
VcrDataRef( const VcrDataRef &copy)
VcrDataRef()
```

dataPtr データに対するポインタ。

data リファレンス値に関する実データ。

copy VcrDataRefのコピー。

type データのタイプ。

doDelete リファレンスがゼロに行く場合にデータを削除すること。

(74)

**doDeref** リファレンスがゼロに行く場合にオブジェクトの引用を解除すること。

リファレンスオブジェクトを作成する。データがVValueに記憶されて提供される場合、データは一定でなければならない。VValueが供給され、そして、タイプがゼロである場合には、タイプは、VValueに記憶されているタイプから初期化される。これは、データのタイプに関するあらゆる別名情報を失う可能性のあることに注意すること。基本的なタイプ（オブジェクトポインタ、ポインタ、ストラクト、等々ではない）に

限り、VValueにおいてサポートされる。タイプがオブジェクトポインタではない場合には、doDerefは無視され、そして、doDeleteがTRUE（真）であり、リファレンスの数がゼロに達する場合には、UxFreeはデータに関してコールされる。タイプがオブジェクトポインタである場合、doDeleteがTRUEであるならば、リファレンスカウントがゼロに到達するとき、クラスインスタンスに関するデストラクタがコールされる。そうでない場合において、doDerefがTRUEであれば、クラスインスタンスの解放方法がコールされる。アーギュメントなしのコンストラクタは、空のリファレンスを作成する。

**Address**

**voic \*Address() const**

**Address**は、ポインタをデータに戻す。コンストラクタ又はPutPointerにパスされたポインタは戻される。リファレンスがVValueからの定数によって初期化される場合には、データに対するポインタも戻されるが、定数に対するリファレンスは、それ自体内の記憶場所にポインタを戻す。従って、リファレンスが消滅した後において、ポインタを使用してはならない。

**Type**

**VTypeData \*Type() const**

**Type**は、データのタイプを戻す。これは、コンストラクタ、又は、PutPointer、又は、PutValueにパスされた値と同じである。

(75)

V a l u e

VValue Value() const

データが定数である場合には、V a l u eは内部V V a l u e記憶を戻す。データがポインタである場合には、無効V V a l u eが戻される。I s C o n t a i n e dがTRUEを戻す場合に限り、この方法がコールされなければならない。

P u t P o i n t e r

```
void PutPointer( void      *dataPntr,
                  VTypeData *type,
                  bool_t    doDelete=FALSE,
                  bool_t    doDelef=FALSE)
```

d a t a P n t r データに対するポインタ。

t y p e データのタイプ。

d o D e l e t e リファレンスがゼロに行く場合に、データを削除すること。

d o D e r e f リファレンスがゼロに行く場合に、オブジェクトの引用を解除すること。

リファレンスオブジェクトをリセットする。タイプがオブジェクトポインタでないならば、d o D e r e fは無視され、そして、d o D e l e t eがTRUEであり、そして、リファレンスの数がゼロに達する場合には、U x F r e eはデータに関してコールされる。タイプがオブジェクトポインタである場合、d o D e l e t eがTRUEであるならば、リファレンスカウントがゼロに達すると、クラスインスタンスに関するデストラクタがコールされる。そうでない場合には、d o D e r e fがTRUEであるならば、クラスインスタンスの解除方法がコールされる。以前にリファレンスされたあらゆるデータは、あたかもこのリファレンスが破壊されてしまったかのように扱われる。

P u t C o n s t a n t

(76)

```
void PutConstant( VV alue      data
                  VTypeData    *type=0)
```

`type` データのタイプ。

`dataPtr` データに対するポインタ。

`doDelete` リファレンスがゼロに行く場合に、データを削除すること。  
。

`doDeref` リファレンスがゼロに行く場合に、オブジェクトの引用が解除される。

リファレンスオブジェクトをリセットする。データは一定でなければならない。タイプがゼロである場合には、タイプは、`VV alue`に記憶されているタイプから初期化される。これは、データのタイプに関するあらゆる別名情報を失う可能性のあることに注意すること。基本的なタイプ（オブジェクトポインタ、ポインタ、ストラクト、等々ではない）に限り、`VV alues`においてサポートされる。以前にリファレンスされたデータは、このリファレンスがあたかも破壊されてしまったかのように扱われる。

`putCopy`

```
void PutCopy(const VcrDataRef &copy)
```

`copy` `VcrDataRef`のコピー。

リファレンスオブジェクトをリセットする。リファレンスの新規な値は、コピーの値と同じである。コピーが`VV alue`ベースの`VcrDataRef`でなかった場合に限り、これは、コピーと同じデータをポイントする。この場合、この`VcrDataRef`は、コピーにおける値の複製を含む。以前にリファレンスされたデータは、このリファレンスがあたかも破壊されてしまったかのように扱われる。

`PutOwnership`

```
void PutOwnership( bool_t      doDelete=FALSE,
                   bool_t      doDeref=FALSE)
```

`doDelete` リファレンスがゼロに行く場合、データを削除すること。

(77)

**d o D e r e f** リファレンスがゼロに行く場合、オブジェクトの引用を解除すること。

データが引用される場合には、**P u t O w n e r s h i p**は所有権情報を変更する。データが**V V a l u e**に記憶される場合には、この方法は効果が無い。この方法は、一般に、オブジェクトまたはデータの所有権を引き継ぐために用いられる。コーラーは、データの寿命が、当該データに関する全てのリファレンスの寿命よりも長いことを確認しなければならない。

**I s C o n t a i n e d**

**bool\_t IsContained() const**

**I s C o n t a i n e d**は、データが、リファレンス内の**V V a l u e**に記憶されている定数であるかどうかを戻す。

**W i l l D e l e t e**

**bool\_t WillDelete() const**

**W i l l D e l e t e**は、当該データに関する全てのリファレンスが消滅する場合に、データが削除されるかどうかを戻す。データが、リファレンス内の**V V a l u e**に記憶されている定数である場合には、これは**T R U E**を戻す。

**W i l l D e r e f**

**bool\_t WillDeref() const**

**bool\_t WillDeref() const**

**W i l l D e r e f**は、当該オブジェクトに関する全てのリファレンスが消滅する場合に、オブジェクトが引用解除されるかどうかを戻す。データが、リファレンス内の**V V a l u e**に記憶されている定数である場合には、これは、**F A L S E**を戻す。

**B a s e** (ベース) クラス

これらのクラスは、全てのクラスレジストリエントリーのためのベースクラスである。それらは、殆どの異なるクラスを通じて必要とされる一般的方法及び特性を提供する。**V c r B a s e**は、全てのクラスレジストリクラスに関する最小公分母を定義し、リファレンスカウント、ヘルプ情報、等々をサポートする。その方法の多くは、純粹に仮想であり、そして、サブクラスにおいて実行される。

(78)

`VcrTopLevel`は、ネームスペースにおいて直接見付けることのできるクラスレジストリアイテムのためのベースクラスであり、クラスレジストリエントリーのバージョンナンバリング及び視点ビュー情報のメンテナン

スのためのサポートを提供する。

### クラスVcrBase

#### `VrPrimary`の共用サブクラス

`VcrBase`は、全てのクラスレジストリエレメント及びタイプ記述に関するベースクラスである。`VcrBase`は、アイテムに関する名前、タイプ、使用、及び、ヘルプ情報に関する方法および記憶装置を提供する。このベースクラスは、その派生クラスにおける使用のためのリファレンスカウントをサポートする。`VerBase`は、直接的にインスタンス生成可能ではない。

#### 保護されたメンバー (Protected Members)

`AcquireChild` 子に関するリファレンスを獲得するためにベースクラスに尋ねる。

`ReleaseChild` 子に関するリファレンスを解除するためにベースクラスに尋ねる。

#### オーバーライド可能な保護されたメンバー (Overridable Protected Members)

`AcquireChildren` 子等に関する内部リファレンスを獲得する。  
。

`ReleaseChildren` 子等に関する内部リファレンスを解除する。  
。

#### 共用メンバー

`AcquireInternal` オブジェクトに関する内部リファレンスを獲得する。

`ReleaseInternal` オブジェクトに関する内部リファレンスを解除する。

`putOwner` オブジェクトの所有者を変更する。

`PutHelp` このアイテムのためにヘルプ情報オブジェクトをセットする

`Tag` このアイテムを識別するタグを戻す。

オーバーライド可能な共用メンバー

`Name` アイテムの名前を戻す。

`Type` アイテムのタイプを戻す。

`Usage` アイテムのための使用フラグを戻す。

`Owner` 所有者にオブジェクトを戻す。

`Help` このアイテムに関するヘルプ情報オブジェクトを戻す。

`ObjectSystem` このアイテムを管理するオブジェクトシステムを戻す。

保護されたデータメンバー

`VTAtomp itsName` アイテムの名前。

`VTUsageCode itsUsage` アイテムの使用フラグ。

`AcquireChild`

```
void AcquireChild( VerBase      *child,
                  VTInternalRefType refType=KVDirectRef)
```

`child` 子に関するリファレンスを獲得するためのチャイルド。

`refType` 子に関するリファレンスを獲得するためのリファレンスのタイプ。

この方法は、オブジェクトの子に関する内部リファレンスを獲得する。例えば、クラスは、その方法、特性、タイプ、等々にリファレンスを直接加える。新規な子がオブジェクトに加えられる時にはいつでも、そして、`AcquireChildren`がコールされる場合にはこの方法がコールされなければならない。子に加えるためのリファレンスタイプの選択を次に示す：

\* 直接内部リファレンスは、子に関してはそれらの所有者別に、そして、それらのタイプオブジェクトに関してはアイテム別に保持される。`AcquireChild (KVDirectRef)` は、内部リファレンス参を直接加える。

\* ループ内部リファレンスは、クラス、ファンクションの関するタイプ及び他

(80)

のタイプ別に保持される。AcquireChild (KVLoopRef) は、ループ内部リファレンスを加える。

ReleaseChild

```
void ReleaseChild( VcrBase      *child,
                   VTInternalRefType refType,
                   bool_t        decRef  =TRUE,
                   bool_t        deleteSelf=TRUE)
```

child 子からリファレンスを解除するためのチャイルド。

refType 子に関するリファレンスを解除するためのリファレンスのタイプ。

decRef リファレンスカウントのデクリメント。

deleteSelf リファレンス参カウントがゼロに到達した場合の自己削除。

ReleaseChildは、オブジェクトの子から内部リファレンスを除去する。子がオブジェクトから解任されるときはいつでも、そして、ReleaseChildrenがコールされた場合に、この方法がコールされなければならない。

AcquireChildren

```
void Acquirechildren()=0
```

この方法は、オブジェクトの全ての子に関する内部リファレンスを獲得する。この方法は、AcquireInternalに限ってコールされる。例えば、クラスは、リファレンスをその方法、特性、タイプ、等々に直接加える。子に加えるためのリファレンスタイプの選択を次に示す：

\* 子に関する直接的な内部リファレンスは所有者別に、そして、それらのタイプオブジェクトに関してはアイテム別に保持される。AcquireChild (KVDirectRef) は、直接内部リファレンスを加える。

\* ループ内部のリファレンスは、クラス、ファンクションに関するタイプ、及び、他のタイプに関するタイプ別に保持される。AcquireChild (KVLoopRef) は、ループ内部リファレンスを加える。

(81)

## ReleaseChildren

```
void ReleaseChildren( bool_t    decRef    =TRUE,
                      bool_t    deleteSelf=TRUE)=0
```

`decRef` リファレンスカウントのデクリメント。

`deleteSelf` リファレンスカウントがゼロに到達した場合の自己削除。

`ReleaseChildren`は、その全ての子から内部リファレンスを解除する。この方法は、`ReleaseInternal`に限ってコールされる。ブーリアン・アーギュメントは、子への`ReleaseChild`コールにパスされなければならない。`deleteSelf`がTRUEであれば、`ReleaseChildren`は、その子に対するあらゆるポインタをNULLによって交換することが出来る。`deleteSelf`がFALSEであるならば、`ReleaseChildRefs`は、その子供に対するポインタを解除してはならない。

## AcquireInternal

```
VRefCount AcquireInternal(
                      VTInternalRefType refType=KVDirectRef)
```

`refType` オブジェクトに関するリファレンスを獲得するためのリファレンスのタイプ。

この方法は、オブジェクトに関する内部リファレンスを獲得する。`AcquireInternal`は、新規に加えられたリファレンスを含むオブジェクトに関する外部および内部リファレンスの全数を戻す。内部リファレンスは、クラスレジストリ階層内において制御されるオブジェクトに関するリファレンスである。外部リファレンスは、クラスレジストリ階層外において制御されるオブジェクトに関するリファレンスである。内部リファレンスはリファレンスカウントにおける循環性を回避するために使われる。この方法は`AcquireChild`に限ってコールされる。リファレンスタイプの選択を次に示す：

\* 外部リファレンスは、クラスレジストリ外のオブザーバによってのみ保持さ

れる。外部リファレンスを加えるために獲得がコールされなければならない。

\* 直接内部リファレンスは、それらの子に関しては所有者別に、そして、それらのタイプオブジェクトに関してはアイテム別に保持される。AcquireInternal (kVDirectRef) は、直接内部リファレンスを加える。

\* ループ内部リファレンスは、クラス、ファンクションに関するタイプ、及び、他のタイプ別に保持される。AcquireInternal (kVLoopRef) は、ループ内部リファレンスを加える。

ReleaseInternal

```

VRefCount ReleaseInternal(
    VTInternalRefType refType    =KVDirectRef,
    bool_t             decRef     =TRUE,
    bool_t             deleteSelf =TRUE)

```

refType オブジェクトに関するリファレンスを獲得するためのリファレンスのタイプ。

decRef リファレンスカウントのデクリメント。

deleteSelf リファレンスカウントがゼロに到達した場合の自己削除。

ReleaseInternalは、オブジェクトに関する内部リファレンスを解除する。戻り値は、残存するオブジェクトに関する外部および内部リファレンスの全数である。ReleaseInternalは、必要に応じて、ReleaseChildRefsをコールする。ReleaseInternalは、循環リファレンスの解決について配慮する。この方法は、ReleaseChildに限ってコールされる。

PutOwner

```
void PuOwner(VcrBase *owner)
```

owner このオブジェクトの所有者

PutOwnerは、このオブジェクトの所有者をセットする。所有者特性は、ルーツタイプ又はクラス定義にまでインスタンスツリーを追跡することを可能

(83)

にするために使われる。この方法は、V c r B a s eの事前決定されたサブクラスによって自動的にコールされる。

H e l p、及び、P u t H e l p

```
VcrHelp *Help()
void PutHelp(VcrHelp *help)
```

h e l p このアイテムに関するヘルプ。

H e l p特性は、このアイテムに関するヘルプシステムオブジェクトを記憶する。ヘルプシステムオブジェクトは、このオブジェクトに関する記述情報、並びに、固有ヘルプシステムからヘルプをディスプレイする能力を提供する。H e l p特性がN U L Lであるならば、このアイテムは一切のヘルプをサポートしない。

N a m e

```
VAtomRef Name()
```

N a m eは、オブジェクトの名前を返す。名前は、V c r B a s eのサブクラスのコンストラクタに対するアーギュメントとして指定された。

T a g

```
VTClassTag Tag() const
```

T a gは、オブジェクトのタグを返す。タグは、このオブジェクトがV c r B a s eのサブクラスのどのタイプであるかを識別する。各サブクラスは、正しい具体化を提供するために、この方法を再定義する。

T y p e

```
Vtypedata *Type()
```

T y p eは、オブジェクトのタイプを返す。デフォルト具体化はN U L Lを返す。この方法は、有意義である場合においては、各々のV c r B a s eサブクラスにおいて再定義されなければならない。すなわち、V C l a s s D a t aは、クラスに関するオブジェクトリファレンスであるタイプを返すべきである。V F u n c t i o n D a t aは、ファンクションの戻りタイプを返すべきである。V P r o p D a t aは、特性のタイプを返すべきである。V c r A r g u m e n t

(84)

はアーギュメントのタイプを戻すべきである。VTypeDataは自身を戻すべきである。

Usage

VTUsageCode Usage()

Usageはオブジェクトの使用フラグを戻す。使用フラグは、VcrBaseのサブクラスのコンストラクタに対するアーギュメントとして指定された。

Owner

VcrBase \*Owner()

Ownerはこのオブジェクトを所有するVcrBaseオブジェクトを戻す。たとえば、VFurictionDataがVClassData::AddMethodを用いてVClassDataに加えられる場合には、VFurictionDataの所有者

はVClassDataである。VcrBaseのサブクラスは、VcrBase::PutOwnerを適当に使用して、所有者をセットする。更に詳細な情報に関してはPutOwnerを参照すること。

ObjectSystem

VTOBJECTSYSTEM objectsSystem()

ObjectSystemは、このアイテムを管理するオブジェクトシステムを戻す。この方法に関するデフォルト具体化はkVNoObjectSystemを戻す。VcrArgumentは、その所有者のオブジェクトシステムを戻すために、この方法を再定義する。

クラスVcrToplevel

VcrBaseの共用サブクラス

VcrToplevelは、ネームスペースにおいて直接見付けることのできるクラスレジストリアイテムに関するベースクラスである。これは、このアイテム、アイテムのバージョン番号、及び、そのフルネームが発見されるビューに関する特定の情報を記憶する。このクラスは、インスタンスを直接生成可能ではない。

(85)

## 共用メンバー

**V i e w s** このアイテムを引用するビューを戻す。

**I n V i e w** このクラスが指定済みのビューツリー内に既に所在するかどうかを戻す。

**A d d V i e w** リファレンスのリストにビューを加える。

**R e m o v e V i e w** リファレンスのリストからビューを除去する。

**U n m a p S e l f** 全てのビューからアイテム及びそのアダプタを除去する。

## オーバーライド可能な共用メンバー

**M a j o r V e r s i o n** このアイテムのメジャバージョンを戻す。

**M i n o r V e r s i o n** このアイテムのマイナバージョンを戻す。

## タイプの定義

<b>VTToplevelRefList</b>	<b>VcrToplevel</b> オブジェクトの <b>VrRefList</b>
<b>VTToplevelRef</b>	<b>VcrToplevel</b> オブジェクトへの <b>VrReference</b>
<b>VTToplevelDict</b>	<b>VcrToplevel</b> オブジェクトの <b>VAtomDictionary</b>

**V i e w s**

**VTViewList Views() const**

**V i e w s** は、現在このアイテムを引用する全てのビューネームスペースを戻す。同一ツリーからの2つのビューネームスペースは、同時に、アイテムを引用することは出来ない。

**I n V i e w**

**bool\_t InView(VViewNameSpace \*node) const**

**n o d e** ビューネームスペースツリーにおけるチェックのためのノード。

**I n V i e w** は、ノードを所有するビュー内のどこかにおいて指定されたアイテムが既に見付けられているかどうかを戻す。この方法は、それがツリー内に既に所在するかどうかをチェックするために使用出来る。

**A d d V i e w**

**void AddView(VViewNameSpace \*node)**

**n o d e** 加えるためのビュー内のノード。

`AddView`は、このアイテムを引用するビューのリストに1つのビューを加える。`node`は、当該アイテムを見付けることの出来るビュー内の実場所である。

`RemoveView`

`void RemoveView(VViewNameSpace *node)`

`node` 除去するためのビュー内のノード。

`RemoveView`は、このアイテムを引用するビューのリストから1つのビューを除去する。`node`は、当該アイテムを見付けることの出来るビュー内の実場所である。

`UnmapSelf`

`void UnmapSelf()`

`UnmapSelf`は、全てのビューから、及び、そのアダプタから、当該アイテムをアンマップする。これは、このアイテムをキャッシュからフラッシュアウトする効果を持つ。これは、制御メモリの消費をヘルプするために、アイテムに関してコールすることが出来る。この方法のこの具体化は、取り付けられた全てのビュー、及び、このアイテムを所有するアダプタに関して`Unmap`をコールする。

`MajorVersion`

`VTVersion MajorVersion()`

`MajorVersion`は、アイテムのメジャーバージョン番号を戻す。デフォルト具体化は0を戻す。この方法は、正当なメジャーバージョン番号を戻すために、サブクラスにおいてオーバーライド可能である。

`MinorVersion`

`VTVersion MajorVersion()`

`MinorVersion`は、アイテムのマイナバージョン番号を戻す。デフォルト具体化は0を戻す。この方法は、正当なマイナバージョン番号を戻すために、サブクラスにおいてオーバーライド可能である。

`Exposed` (露出) クラス

これらは、クラスレジストリのユーザーに露出されているクラスである。これらのクラスは、クラスレジストリエントリのための構造を定義する。VFunctionData、VPropData、VClassData、VAdapterNameSpace、及び、VViewNameSpaceは、各オブジェクトシステムの特種サポートを提供することができるようにサブクラス分類することを意図した抽象クラスである。VcrArgument、VExceptionData、及び、VInstanceDataは、直接インスタンス化可能であるか、又は、特定の目的のためにサブクラス化可能である。VClassRegistryは直接インスタンス化可能であり、サブクラス化されてはならない。露出されたクラスの集合は、タイプ管理セクションにおいて定義されるVTypeDataを含む。

#### クラスVcrArgument

VcrBaseの共用サブクラス

VcrArgumentインスタンスは、クラスレジストリ記述された方法に関するアーギュメントを指定する。

共用メンバー

VcrArgument VcrArgumentオブジェクトを作成する。

Default アーギュメントのデフォルト値を戻す。

PutDefault アーギュメントのデフォルト値をセットすること。

タイプの定義

VTAArgumentList VcrArgumentオブジェクトのVArray。

VcrArgumentコンストラクタ

```
VcrArgument( VAtomRef    name,
              VTypeData   *type,
              VTUsageCode ucodes=0)
```

name アーギュメントの名前。

type アーギュメントのタイプ。

ucodes アーギュメント用使用フラグ（該当する場合）。

アーギュメントオブジェクトを作成する。このオブジェクトは、一般にその作成の後において、`AddArgument`を用いて、`VFunctionData`に加えられる。

デフォルト、及び、`PutDefault`

```
const VcrDataRef &Default()
void PutDefault(VcrDataRef)
```

デフォルト特性はアーギュメントに関するデフォルト値を記憶する。方法がコールされた場合にアーギュメントが指定されない場合には、この値が使用される。アーギュメントタイプが基本タイプである場合に限り、デフォルト値が用いられる。

### クラスVFunctionData

`VcrTopLevel`の共用サブクラス

`VFunctionData`は、宣言されたファンクション（或いは方法）を示す抽象クラスである。これは、その名前、アーギュメント、戻りタイプ、及び、コール可能なエントリーポイントを含む。このオブジェクトは、クラスレジストリ、または、タイプマネージャー記述において使用可能である。又は、サブクラスは、直接ファンクションコール用の内部サポートを提供しないアプリケーションにおける支援ファンクションコール明白な目的のために作成可能である。方法はオブジェクトポインタである隠された第1のアーギュメントを持たねばならないという点で、方法とファンクションは異なる。このアーギュメントは、総称オブジェクトポインタとして宣言される。従って、方法（`VcrCall`を用いる）を実行する以前に、第1のアーギュメントに関して正しいタイプのチェックを実施することは、コーラーの自由裁量に従う。方法のために隠されたアーギュメントを加えることは、`VFunctionData`のサブクラスの責任である。

。

共用メンバー

`VFunctionData` `VFunctionData`オブジェクトを作成する。

`Equal` このファンクション定義が指定されたファンクションの定義と同

(89)

じであるかどうかを決定する。

オーバーライド可能な共用メンバー

`LockAccess` このスレッドがファンクションをコールできるかどうかをチェックする。

`UnlockAccess` コールファンクションに関するこのスレッドのロックを解く。

`FunctionType` このタイプのファンクションに対するポインタを表すタイプを戻す。

`IsMethod` これが方法であるかどうかを戻す。

`Arguments` ファンクションのためのアーギュメントのリストを戻す。

`Exceptions` ファンクションが作ることのできる例外のリストを戻す。

`EntryPoint` ファンクションのためのコール可能なエントリポイントを戻す。

`ValidCall` 前のコールが成功したかどうかを戻す。

タイプの定義

`VTFunctionList` `VFunctionData` オブジェクトの `VArray`。

`VTFunctionRefList` `VFunctionData` オブジェクトの `VrRefList`。

`VTFunctionRef` `VFunctionData` オブジェクトに対する `VrReference`。

`VTFunctionDict` `VFunctionData` オブジェクトの `VAtomDictionary`。

`VFunctionData` コンストラクタ

```
VFunctionData(  VAtomRef    name,
                 VTUsageDode  ucodes=0)
```

(90)

`name` ファンクションの名前。

`ucodes` ファンクションに関する使用フラグ（該当する場合）。

ファンクション記述を作る。ファンクションの定義が完了した後において、クラスまたはネームスペースに記憶可能である。

`Equal`

`bool_t Equal(VFunctionData *func)`

`func` 比較するためのファンクション。

`Equal` は、2つのファンクション記述を比較し、それらが等しければ真を返す。2つのファンクションが等しいためには、名前、戻しのタイプ、使用コード、アーギュメントの個数、及び、ファンクション記述の例外が同じでなければならない。更に、全てのアーギュメントの名前、タイプ、及び、使用コードが等しくなければならない。ヘルプ情報はチェックされない。2つのファンクションポインタのタイプが同等のタイプであるかどうかを決定する場合には、この方法がタイプマネージャーによってコールされる。

`LockAccess`

`bool_t LockAccess(bool_t wait)`

`wait` ロックが解除されることを待つ。

他のスレッドがファンクションを同時にコールしないように、`LockAccess` は `VFunctionData` をロックする。`VFunctionData` に関して `KVUageThreadSafe` が指定された場合には、`LockAccess` は必ず `TRUE` を返す。`wait` が `TRUE` であり、そして、ファンクションが現在ロックされている場合には、`LockAccess` はファンクションが解錠されることを待つ。そうでない場合には、`LockAccess` は `FALSE` を返す。`LockAccess` が `TRUE` を戻した場合には、`VFunctionData::Unlockcall` は後でコールされなければならない。

`UnlockAccess`

`void UnlockAccess()`

(91)

他のスレッドが当該ファンクションをコールすることができるように、`UnlockAccess`は`VFunctionData`をアンロックする。`VFunctionData`に関して`kVUsageThreadsafe`が指定されている場合には、`UnlockAccess`は何もしない。

#### `FunctionType`

`VTypeData *FunctionType()=0`

このサインを持ったファンクションに対するポインタを表すタイプを戻す。

#### `IsMethod`

`bool_t IsMethod()=0`

このオブジェクトがクラスの方法である（すなわち、タイプオブジェクトの隠された第1アーギュメントを持つ）かどうか、又は、それがファンクションであるかどうかを戻す。

#### `Arguments`

`VTArgumentList Arguments()=0`

このファンクションに関するアーギュメントのリストを戻す。ファンクションが直接または間接的にクラスレジストリに入れられた後で、リストは、追加されるか又は除去されるアイテムを一切所有しない。

#### `Exceptions`

`VTEExceptionList Exceptions()=0`

このファンクションが作ることの出来る例外のリストを戻す。このリストは、コールコードによって修正されるべきでない。言語は、この情報を使う必要が無い。ファンクションが`Call`方法によってコールされる場合には、当該例外に関する必要な全ての情報が戻される。この方法は、主として情報的な目的のために使われる。リストは、ファンクションが直接または間接的にクラスレジストリに入れられた後で、それに追加またはそれから除去されるべきアイテムは一切所有しない。

#### `EntryPoint`

```
VerCallException *ValidCall(
                                VFunctionData *funcData,
                                unsigned paramCount,
```

(93)

VTypeData	**paramTypes,
void	**params,
VcrDataRef	&result)=0

f u n c D a t a 当該コールに関するファンクション記述。

p a r a m C o u n t パラメータの数。

p a r a m T y p e s パラメータのタイプ。

p a r a m s パラメータに対するポインタの配列。

r e s u l t 当該ファンクションの戻り値に対するポインタ。

V a l i d C a l l は、当該ファンクションに対する以前のコールが正当な結果を生成したかどうかをチェックする。この方法は、戻り値、及び、例外情報に関する出力またはイン／アウトパラメータをチェックするために、サブクラスにおいて再実行されなければならない。

#### クラスVPropData

V c r B a s e の共用サブクラス

V P r o p D a t a は、クラスの特性を記述する抽象クラスである。特性は、ゲットアンドセットアクセッサ方式によってアクセスされるクラスの名前付き属性である。特性は、「ゲット」及び「セット」の2つのアクセッサ方式の少なくとも一方を定義しなければならない。アクセッサ方式は、多重アーギュメントを持つことができる。例えば、特性が実際に収集内の特定の場所へアクセスするためにインデックスを必要とする他のオブジェクトの収集である場合に、これは、有用である。

共用メンバー

V P r o p D a t a V P r o p D a t a オブジェクトを作成する。

オーバーライド可能な共用メンバー

L o c k A c c e s s このスレッドがファンクションをコールすることができるかどうかをチェックする。

U n l o c k A c c e s s コールファンクションに関するこのスレッドのロックを解錠する。

G e t M e t h o d 特性のゲット方式を戻す。

(94)

`SetMethod` 特性のセット方式を戻す。

タイプの定義

`VTPropList`

`VPropData` オブジェクトの `VArray`

`VPropData` コンストラクタ

```
VPropData( VAtomRef name,
            VTUsageCode ucodes=0)
```

`name` 特性の名前。

`ucodes` 特性に関する使用フラグ（該当する場合）。

特性記述を作成する。アクセッサ方式の名前は、`__get__` 又は `__set__` のいずれかによって先行される特性の名前でなくてはならない。このネーミングコンベンション（CORBA C言語により義務付けられる）は、多重クラスレジストリによって作動可能にされるプログラミング言語を堅実に横断して特性にアクセスすることを可能にする。

`LockAccess`

```
bool_t LockAccess(bool_t wait)
```

`wait` 解錠されることを待つこと。

他のスレッドが特性に同時にアクセス出来ないようにするために、`LockAccess` は `VPropData` をロックする。`KVUsageThreadSafe` が `VPropData` に関して指定されている場合には、`LockAccess` は常に `TRUE` を戻す。`wait` が `TRUE` であり、そして、ファンクションが現在ロックされている場合には、`LockAccess` は、ファンクションが解除されることを待つ。そうでない場合には、`LockAccess` は `FALSE` を戻す。`LockAccess` が `TRUE` を戻す場合には、`VPropData::UnlockCall` は後でコールされなければならない。デフォルトとしての実行は、戻す以前に、ゲットアクセッサ及びセットアクセッサの両方をロックする。

`UnlockAccess`

```
void UnlockAccess()
```

他のスレッドが特性にアクセス可能であるように、UnlockAccessはVPropDataを解錠する。VPropDataに関してKVUsageThreadsafeが指定されている場合には、UnlockAccessは何もしない。

### GetMethod

VFunctionData \*GetMethod()=0

GetMethodは、特性に関する検索アクセッサ方式を戻す。この方法は、コンストラクタの一部としてインストールされた。戻りがNULLである場合には、特性は読み取り可能でない。これは、最初の呼出しの後において、必ず、同一値を戻すべきである。

### SetMethod

VFunctionData \*SetMethod()=0

SetMethodは、特性に関する記憶アクセッサ方式を戻す。この方法は、コンス

トラクタの一部として特性にインストールされた。戻しがNULLである場合には、特性は記入可能でない。これは最初の呼出しの後において、必ず、同一値を戻すべきである。

### クラスVInstanceData

#### VcrTopLevelの共用サブクラス

VInstanceDataは、定数、変数、または、クラスの名前付きインスタンスを記述する。名前付きインスタンスは、一般的に利用可能なインスタンスであるか、又は、或るクラスに特有であるように、クラスレジストリにインストール可能である。VInstanceDataは、必要に応じて、サブクラスに分類されても差し支えない。使用フラグは、インスタンスが読み専用、又は、読み書き、又は、書き専用のいずれであるかを指定する。定数は、読み専用インスタンスとして記述される。

#### 共用メンバー

VInstanceData VInstanceDataオブジェクトを作

(96)

成する。

オーバーライド可能な共用メンバー

V a l u e インスタンスの値を戻す。

保護されたメンバー

P u t V a l u e インスタンスの値をセットする。

タイプの定義

VTInstanceList	VInstanceData オブジェクトのVArray
VTInstanceRefList	VInstanceData オブジェクトの VrRefList
VTInstanceRef	VInstanceData オブジェクトへの VrReference
VTInstanceDict	VInstanceData オブジェクトの VAtomDictionary

V i n s t a n c e D a t a コンストラクタ

```
VInstanceData( VAtomRef    name,
               VcrDataRef  value,
               VTUsageCode  ucodes=0)
```

n a m e インスタンスの名前。

v a l u e インスタンスの値。

u c o d e s インスタンスのための使用フラグ（該当する場合）。

名前付きインスタンスの記述を作成する。

P u t V a l u e

```
void PutValue(VcrDataRef value)
```

v a l u e インスタンスの値。

名前付きインスタンスの値をセットする。これは、ベースクラスの初期化の後において定数の値をセットすることが必要な V I n s t a n c e D a t a のサブクラスからコールされることを意図したものである。

V a l u e

```
const VcrDataRef &Value()
```

名前付きインスタンスの値を戻す。

クラス V E x c e p t i o n D a t a

## VcrTopLevelの共用サブクラス

VExceptionDataは、ユーザー定義の例外を記述する。例外は、一般的に利用可能な例外であるか、又は、クラスに対して指定されるようにクラスレジストリにインストールされても差し支えない。例外は、それらの名前によって区別される。VExceptionDataは、必要に応じて、サブクラスに分類されても差し支えない。

### 共用メンバー

VExceptionData VExceptionDataオブジェクトを作成する。

### 保護されたメンバー

PutType 例外のタイプをセットする。

### タイプの定義

VTEExceptionList	VExceptionData オブジェクトの VArray
VTEExceptionRefList	VExceptionData オブジェクトの VrRefList
VTEExceptionRef	VExceptionData オブジェクトへの VrReference
VTEExceptionDict	VExceptionData オブジェクトの VAtomDictionary

## VExceptionDataコンストラクタ

```
VExceptionData(    VAtomRef    name,
                   VTypeData    *tape,
                   VTUsageCodes  ucodes=0)
```

name 例外の名前。

type 例外のタイプ。

ucodes 例外のための使用フラグ（該当する場合）。

ユーザー定義例外の記述を作成する。

### PutType

```
void PutType(VTypeData *type)
```

type 例外のタイプ。

ユーザー定義例外のタイプをセットする。これは、ベースクラスの初期化の後

において例外のタイプをセットすることが必要な `VExceptionData` のサブクラスからコールされることを意図したものである。例外のタイプは、それが最初に検索された後においては変更されてはならない。

### クラス `VClassData`

`VcrTopLevel` の共用サブクラス

`VClassData` は、クラスレジストリに対するオブジェクトのクラスを記述する抽象クラスである。その定義は、クラスによって所有される特性、方法、タイプ等の集合について質問するためのサポートを提供する。方法は、コンストラクタ方法、デストラクタ方法、及び、リファレンスカウント方法を含む。クラスに関して記述する場合には、これら全ての方法は定義されない状態にしても差し支えない。このクラスは直接サブクラス化しても差し支えなく、或いは、`VcrCodedClass` は、オブジェクトシステムアダプタの代りにサブシステム化しても差し支えない。

共用メンバー

`VClassData VClassData` オブジェクトを作成する。

`Notifyviews` クラスが変化したことをビューに通告する。

オーバーライド可能な共用メンバー

`BaseClass` 記述されたクラスの第1ベースクラスを戻す。

`BaseClasses` 全てのベースクラスのリストを戻す。

`Constructor` クラスのコンストラクタ記述を戻す。

`Duplicator` クラスのコピーコンストラクタ記述を戻す。

`Destructor` クラスのデストラクタ記述を戻す。

`AcquireMethod` クラスの「獲得」方法の記述を戻す。

`ReleaseMethod` クラスの「解放」方法の記述を戻す。

`Methods` クラスの全ての方法のリスト、或いは、指定された名前を持つ全ての方法のリストを戻す。

`Properties` クラスの全ての特性のリストを戻す。

`Types` クラスの全てのタイプのリストを戻す。

(99)

`I n s t a n c e s` クラスにおいて定義された全てのインスタンスのリストを返す。

`E x c e p t i o n s` クラスにおける全ての例外のリストを返す。

`M e t h o d` 指定された名前の方を返す。

`P r o p e r t y` 指定された名前の特性を返す。

`D e f a u l t P r o p e r t y` 記述されたクラスのデフォルト特性を返す。  
。

`T y p e` 指定された名前のタイプを返す。

`I n s t a n c e` 指定された名前のインスタンスを返す。

`E x c e p t i o n` 指定された名前の例外を返す。

`C a s t T o D i r e c t B a s e` オブジェクトインスタンスを直接ベースクラスへキャストする。

`C a s t T o B a s e` オブジェクトインスタンスをあらゆるベースクラスインスタンスへキャストする。

`C a s t F r o m D i r e c t B a s e` 直接ベースクラスからオブジェクトインスタンスをキャストする。

`C a s t F r o m B a s e` あらゆるベースクラスからこのクラスへオブジェクトインスタンスをキャストする。

`S u b c l a s s O f O b j e c t` 指定されたオブジェクトがこのクラスのサブクラスである場合に、指定されたオブジェクトのクラスを返す。

`S u o c l a s s O f` この方法に対してアーギュメントであるクラスのサブクラスであるかどうかを決定すること。

`S u p e r c l a s s O f` この方法に対してアーギュメントであるクラスのスーパークラスであるかどうかを決定すること。

タイプの定義

(100)

VTClassList	VClassData オブジェクトの VArray
VTClassRefList	VClassData オブジェクトの VrRefList
VTClassRef	VClassData オブジェクトへの VrReference
VTClassDict	VClassData オブジェクトの VAtomDictionary

## VClassData コンストラクタ

```
VClassData(    VAtomRef    name,
               VTUsageCode  usage)
```

`name` クラスの名前

`usage` クラスに関する使用フラグ。

クラスの記述を作成する。クラスは、単独で継承されるか、または、複合継承されたベースクラスであっても差し支えない。

クラスのユーザーは、その有効範囲に紹介されたオブジェクトの破壊のために次の規則に従うべきである：

- \* オブジェクトがコンストラクタによって作成された場合には、ユーザーは、適宜、デストラクタ（定義されている場合）をコールしなければならない。
- \* オブジェクトが方法またはファンクションコールによって導入され、次に、オブジェクトを導入したアーギュメント／方法に関する使用フラグがフラグKVUsageCallerIsOwnerを含む場合には、ユーザーは、適宜、デストラクタをコールしなければならない。
- \* そうでない場合には、オブジェクトが導入された場合、AcquireMethodがコールされなければならない、そして、オブジェクトが処理されなければならない場合には、定義済みであれば、ReleaseMethodがコールされなければならない。

これらの方法（コンストラクタ、デストラクタ、AcquireMethod、ReleaseMethod）は、必ずしもオブジェクト自身によって実行されることが必要であるとは限らず、クラスレジストリへの統合を容易にするよに設計された利便なファンクションを用いても差し支えないことに注意されたい。コンストラクタは、任意のアーギュメントを所持しても差し支えないが、デストラ

(101)

クタ、`AcquireMethod`、及び、`ReleaseMethod`はアーギュメントを所持してはならず、或いは、全てのアーギュメントはデフォルト値を持たなければならない。

### `NotifyViews`

```
void NotifyViews(    short    nummeths,
                    short    numprops,
                    short    numtypes,
                    short    numinsts,
                    short    numexcepts)
```

`nummeths` クラスにおける新規な方法の数。

`numprops` クラスにおける新規な特性の数。

`numtypes` クラスにおける新規なタイプの数。

`numinsts` クラスにおける新規なインスタンスの数。

`numexcepts` クラスにおける新規な例外の数。

`NotifyViews`は、クラスの全てのビューに、その定義が変わったことを通告する。クラスが変わった場合にはいつでも、これがコールされなければならない、そして、それに関するビューは既に存在する。更に詳細には、ビューには、方法、特性、タイプ、インスタンス、または、例外の完全なリストの検索が終了した後で修正が発生したかどうかだけを通告する必要がある。質問することを目的として、方法、特性、等々に関する何等かの特定の情報をビューが記憶しつつある場合に、ビューに通告する必要がある。

### `BaseClass`

```
VClassData *BaseClass()=0
```

`BaseClass`は、クラスがベースクラスであるならば`NUL L`を返し、このクラスが単独で継承される場合にはベースクラスを返し、又は、このクラスが多重的に継承される場合には、コンストラクタに供給された第1ベースクラスを返す。

### `BaseClasses`

```
VTClassList BaseClasses()=0
```

(102)

`BaseClasses`は、クラスがベースクラスであるならば空の`VArray`を返し、このクラスが単独で継承される場合にはベースクラスを含む`VArray`を返し、又は、このクラスが多重的に継承される場合にはコンストラクタに供給されたベースクラスのアレイを返す。このアレイは、コーラーによって修正されてはならない。

### `Constructor`

`VFunctionData *Constructor()=0`

`Constructor`は、記述されたクラスに関するコンストラクタを返す。コンストラクタによって戻されたファンクションは、オブジェクトに関するC++コンストラクタと同じではない。クラスレジストリによって定義されたコンストラクタは、オブジェクトの割当てに責任があり、そして、新規なオブジェクトを戻さねばならない。コンストラクタに関するクラスレジストリ記述は総称オブジェクト戻しタイプを持つ（タイプコード`KVTypeObjectPointer`）。指定されたコンストラクタが無い場合もあり得る。この場合には、コンストラクタは`NULL`を返す。この方法によって戻された`VFunctionData`は、オブジェクトインスタンスに関する隠された第1アーギュメントを持たない。従って、この`VFunctionData`は、方法であるとみなしてはならない。

### `Duplicator`

`VFunctionData *Duplicator()=0`

`Duplicator`は、記述されたクラスに関するコピーコンストラクタを返す。`Duplicator`によって戻されたファンクションは、オブジェクトに関するC++コピーコンストラクタと同じではない。クラスレジストリによって定義されたコピーコンストラクタは、オブジェクトの割当てに関して責任を負い、そして、新規なオブジェクトを戻さなければならない。コピーコンストラクタに関するクラスレジストリ記述は、総称オブジェクトリターンタイプを持つ（タイプコード`KVTypeObjectPointer`）。指定されたコピーコンストラクタが無い場合もあり得る。この場合、`Copy`はN

U L Lを戻す。この方法によって戻されたV F u n c t i o n D a t aは、コピーに対するオブジェクトインスタンスに関する隠された第1アーギュメントを持つ。従って、このV F u n c t i o n D a t aは方法であり、そして、コンストラクタと同様ではない。

### D e s t r u c t o r

VFunctionData \*Destructor()=0

D e s t r u c t o rは、記述されたクラスに関するデストラクタを戻す。この方法は、アーギュメントを持ってはならないか、或いは、全てのアーギュメントに対してデフォルト値を持たなければならない。デストラクタによって戻された方法は、オブジェクトインスタンスに割当てられたメモリーの実自由を実施することが方法に責任があるという点において、C++デストラクタと同じではない。指定されたデストラクタが存在しないこともあり得る。この場合、デストラクタはN U L Lを戻す。デストラクタからの戻しは方法でなければならない。

### A c q u i r e M e t h o d

VFunctionData \*AcquireMethod()=0

A c q u i r e M e t h o dは、クラスのインスタンスにリファレンスを加えるために用いられる方法を戻す。この方法はアーギュメントを持ってはならず、又は、全てのアーギュメントはデフォルト値を持たなければならない。A c q u i r e M e t h o dの戻しは同様にN U L Lであっても差し支えない。A c q u i r e M e t h o dからの戻しは方法でなければならない。オブジェクトシステムがリファレンスカウントを継承的にサポートしない場合には、クラスレジストリは、ユーティリティクラスを用いて、インスタンスに基づいたリファレンスカウントに対してサポートを提供することが出来る。

### R e l e a s e M e t h o d

VFunctionData \*ReleaseMethod()=0

R e l e a s e M e t h o dは、クラスのインスタンスからリファレンスを除去するために使われる方法を戻す。この方法はアーギュメントを持ってはならず、又は、全てのアーギュメントはデフォルト値を持たなければならない。R e l e a s e M e t h o dの戻しは同様にN U L Lであっても差し支えない。R e l

(104)

`easeMethod`からの戻しは方法でなければならない。

## Methods

```

VTFunclist Methods()=0
VTFunclist Methods(  VAtomRef    name,
                     VTSearchCase searchas,

                     void          *object=0)=0

```

`name` 探索するための方法の名前。

`searchas` 探索はケース依存またはケース非依存のいずれかである。

`object` 動的探索に使用するオブジェクト

`Methods`は、2つのオーバーロードされた定義を持つ。アーギュメントなしのバージョンがコールされた場合には、当該クラスにおける事前に定義済みの全ての方法のリストが戻される。戻されたリストは修正してはならない。このリストは動的に利用可能な方法は一切含まない。ただし、`VCClassData`のサブクラスによって方法の内部リストにこの種の方法が加えられていた場合にはこの限りでない。この方法の第2バージョンは、所定の名前をもつオーバーロードされた全ての方法を含む。オブジェクトが指定されている場合には、`VCClassData`のサブクラスは、方法の動的ルックアップを遂行するためにこれを使用しても差し支えない。

## Properties

```

VTPropList Properties()=0

```

`Properties`は、クラス記述において定義された全ての特性のリストを戻す。このリストはコーラーによって修正されてはならない。このリストは、動的に利用可能な特性は一切含まない。ただし、`VCClassData`のサブクラスによって、特性の内部リストにこの種の特性が加えられていた場合はこの限りでない。

## Types

```

VTTypeList Types()=0

```

`Types`は、事前に定義済みのタイプによって定義された全てのクラス記述のリストを戻す。このリストは、コーラーによって修正されてはならない。この

(105)

リストは、動的に利用可能なタイプは一切含まない。ただし、`VClassData`のサブクラスによって、特性の内部リストにこの種のタイプが加えられていた場合はこの限りでない。

### Instances

`VTInstanceList Instances()=0`

`Instances`は、クラス記述において定義された全ての名前付きインスタンスのリストを返す。これらは、クラスのインスタンスではあり得ず、クラスによって使われる名前付き定数であることに注意されたい。このリストは、コーラーによって修正されてはならない。このリストは、動的に利用可能なインスタンスは一切含まない。ただし、`VClassData`のサブクラスによって特性の内部リストに加えられていた場合にはこの限り出ない。

### Exceptions

`VTExceptionList Exceptions()=0`

`Exceptions`は、クラス記述において定義された事前に定義済みの全ての例外のリストを返す。このリストは、コーラーによって修正されてはならない。このリストは、動的に利用可能な一切の例外を含まない。ただし、この種の例外が、`VClassData`のサブクラスによって特性の内部リストに加えられていた場合にはこの限りでない。

### Method

`VFunctionData *Method( VAtomRef name,  
VTSearchCase searchas,  
void *object=0)=0`

`name` 探索するための方法の名前

`searchas` 探索は、ケース依存またはケース非依存のどちらかである

。

`object` 動的探索のために使用するオブジェクト

`Method`は、所定の名前の利用可能な第1の方法を発見する。オブジェクトが指定されている場合には、`VClassData`のサブクラスは、方法の動

的ルックアップを遂行するためにこれを使用しても差し支えない。

## Property

```
VPropData *Property( VAtomRef name,
                      VTSearchCase searchas,
                      void *object=0)=0
```

`name` 探索するための特性の名前

`searchas` 探索は、ケース依存またはケース非依存のどちらかである

。

`object` 動的探索のために使用するオブジェクト

`Property`は、クラス記述における所定の名前の特性を発見する。オブジェクトが指定されている場合には、`VClassData`のサブクラスは、特性の動的ルックアップを遂行するためにこれを使用しても差し支えない。方法とは異り、オーバーロードされた特性の概念はない。

## DefaultProperty

```
VPropData *DefaultProperty(void *object=0)=0
```

`object` 動的探索のために使用するオブジェクト

`DefaultProperty`は、クラス記述においてデフォルト特性を発見する。これは、`KVUsageDefaultProperty`使用フラグによって指定された特性である。デフォルト特性のコンセプトは、例えば`Basic`のような言語においてサ

ポートされる。オブジェクトが指定されている場合には、`VClassData`のサブクラスは、特性の動的ルックアップを遂行するためにこれを使用しても差し支えない。

## Type

```
VTypeData *Type( VAtomRef name,
                  VTSearchCase searchas,
                  void *object=0)=0
```

`name` 探索するためのタイプの名前。

(107)

`searchas` 探索は、ケース依存またはケース非依存のどちらかである。

`object` 動的探索のために使用するオブジェクト。

`Type`は、クラス記述において所定の名前のタイプを発見する。オブジェクトが指定されている場合には、`VClassData`のサブクラスは、タイプの動的ルックアップを遂行するためにこれを使用しても差し支えない。

`Instance`

```
VInstanceData *Instance( VAtomRef    name,
                          VTSearchCase searchas,
                          void          *object=0)=0
```

`name` 他なくするためのインスタンスの名前

`searchas` 探索は、ケース依存またはケース非依存のどちらかである。

`object` 動的探索のために使用するオブジェクト。

`Instance`は、クラス記述において所定の名前のインスタンスオブジェクトを発見する。オブジェクトが指定されている場合には、`VClassData`のサブクラスは、インスタンスの動的ルックアップを遂行するためにこれを使用しても差し支えない。これはクラスのインスタンスであり得ないが、クラスによって使用される名前付き定数であることに注意されたい。

`Exception`

```
VExceptionData *Exception( VAtomRef    name,
                            VTSearchCase searchas,
                            void          *obj=0)=0
```

`name` 探索するための例外の名前

`searchas` 探索は、ケース依存またはケース非依存のどちらかである。

`object` 動的探索のために使用するオブジェクト。

`Exception`は、クラス記述において所定の名前の例外を発見する。オブジェクトが指定されている場合には、`VClassData`のサブクラスは、

## 例外の動的ルック

アップを遂行するためにこれを使用しても差し支えない。

`CastToDirectBase`

```
void *CastToDirectBase(   VClassData   *base,
                          void          *object)
```

`base` キャストするためのベースクラス。

`object` キャストするためのオブジェクトインスタンスへのポインタ。

`CastToDirectBase`は、指定されたベースクラスに対してコールする方法において直接使用できる新規なオブジェクトポインタを戻す。ベースが直接的なベースクラスでない限り、デフォルト実施はリターンオブジェクトを戻す。デフォルト実施は、大抵のオブジェクトシステムにとって充分である。ただし、多重ベースクラスを有するC++の場合には、各ベースクラスに対して異なるオブジェクトポインタを戻さねばならない。ベースクラスのリストにベースクラスが発見されない場合には、この方法はNUL Lを戻さねばならない。

`CastToBase`

```
void *CastToBase(   VClassData   *base,
                   void          *object)
```

`base` キャストするためのベースクラス。

`object` キャストするためのオブジェクトインスタンスに対するポインタ。

`CastToBase`は、指定されたベースクラスに対してコールする方法において直接使用することの出来る新規なオブジェクトポインタを戻す。指定されたベースクラスは、階層のどこに所在しても差し支えない。デフォルト実施は、ベースクラスへの通路を発見し、そして、階層の各レベルに対して`CastToDirectBase`をコールする。デフォルト実施は、更に効率的であるためにアダプタによって定義されたサブクラスによって置き換えることができる。ベースクラスの階層においてベースクラスが発見されない場合には、この方法はNUL Lを戻さねばならない。この方法は、指定されたベースクラスへの通路を発

見するために、ツリーの前順走査を遂行する。

`CastFromDirectBase`

```
void *CastFromDirectBase( VClassData    *base,
                          void          *object,
                          bool_t        safe)
```

`base` そこからキャストするためのベースクラス。

`object` キャストするためのオブジェクトインスタンスに対するポインタ。

`safe` タイプセーフキャストを実施する。

`CastFromDirectBase`は、このクラスに対してコールする方法において直接使用することの出来る新規なオブジェクトポインタを返す。ベースが直接ベースクラスでない限り、デフォルト実施はオブジェクトを返す。デフォルト実施は、大抵のオブジェクトシステムに対して充分である。ただし、多重ベースクラスに関するC++の場合には、異なるオブジェクトポインタが各ベースクラスインスタンスを表す。ベースクラスのリストにおいて当該ベースクラスが発見されない場合には、この方法はNULLを返さねばならない。セーフがTRUEである場合には、タイプセーフキャストとしてキャストが実施できる場合に限り、この方法は非NULLを返さねばならない。セーフがTRUEである場合には、デフォルト実施はNULLを返す。

`CastFromBase`

```
void *CastFromBase( VClassData    *base,
                   void          *Object,
                   bool_t        safe)
```

`base` そこからキャストするためのベースクラス。

`object` キャストするためのオブジェクトインスタンスに対するポインタ。

`safe` タイプセーフキャストを実施する。

`CastFromBase`は、このクラスに対してコールする方法において直

接使用できる新規なオブジェクトポインタを戻す。指定されたベースクラスは、階層のどこに所在しても差し支えない。デフォルトとしての実施は、ベースクラスへの通路を発見し、そして、階層の各レベルに対して `CastFromDirectBase` をコールする。デフォルトとしての実施は、更に効率的であるために、アダプタによって定義されたサブクラスによって置き換えられても差し支えない。ベースクラスがベースクラスの階層において発見されない場合には、この方法は `NULL` を戻さねばならない。セーフが `TRUE` である場合には、タイプセーフキャストとしてキャストが実施出来る場合に限り、この方法は、非 `NULL` を戻さねばならない。セーフが `TRUE` である場合には、デフォルトとしての実施は `NULL` を戻す。この方法は、指定されたベースクラスへの通路を発見するために、ツリーの前順走査を実施する。

`SubclassOfObject`

`VClassData *SubclassOfObject(void **object)`  
**object** (Input&Return) A pointer to the object to narrow.

**object** (入力および戻し) 狭くするためのオブジェクトに対するポインタ

オブジェクトが記述されたクラスのインスタンスであるか、或いは、記述されたクラスのサブクラスのインスタンスであると仮定した場合、`SubclassOfObject` は、オブジェクトが所属するサブクラス記述を戻すことによってオブジェクトを狭くする。

サブクラスが決定出来ない場合には、戻しは、現行クラス記述ポインタでなくてはならない。場合によっては、他のポインタによって当該オブジェクトを更に能率的に表わすことが可能であることが、`VClassData` の実施によって決定されることもあり得る。この場合、インプットオブジェクトポインタは修正されても差し支えない。`SubclassOfObject` のコーラーは、クラスレジストリの方法およびオブジェクト自体の方法に対する後続する全てのコールにおいて、修正されたオブジェクトポインタを使用しなければならない。

`SubclassOf`

(111)

```

bool_t SbuclassOf(VClassData *other)
other      Class to use in hierarchy check.

```

other 階層チェックにおいて用いられるクラス。

SbuclassOfは、この記述されたクラスが他のオブジェクトによって記述されたクラスのサブクラスであるかどうかを戻す。

```

SuperclassOf

```

```

bool_t SuperclassOf(VClassData *other)

```

other 階層チェックにおいて用いられるクラス。

SuperclassOfは、この記述されたクラスが他のオブジェクトによって記述されたクラスのスーパークラスであるかどうかを戻す。

### クラスVAdapterNameSpace

VcrTopLevelの共用サブクラス

VAdapterNameSpaceは、各オブジェクトシステムアダプタによってサブクラス化されるべき抽象クラスである。このクラスは、オブジェクトシステムとクラスレジストリとの間のインタフェースを提供する。オブジェクトシステムアダプタは、クラスレジストリオブジェクトをサブクラス化することにより、アイテム記述に関する固有フォーマットと、例えばクラス及び方法のようなアイテムのクラスレジストリ記述との間のインタフェースを提供しなければならない。アダプタは、ネームスペース階層を、その固有階層にできる限り近いクラスレジストリに提供しなければならない。

共用メンバー

VAdapterNameSpace アダプタネームスペースオブジェクトを作成する。

オーバーライド可能な共用メンバー

Subspace 指定された名前のネームスペースを戻す。

Class 指定された名前のクラスを戻す。

Function 指定された名前のファンクションを戻す。

Type 指定された名前のタイプを戻す。

(112)

`I n s t a n c e` 指定されたインスタンスを戻す。

`E x c e p t i o n` 指定された名前の例外を戻す。

`C l a s s e s` 指定された名前のクラスを戻す。

`F u n c t i o n s` 指定された名前のファンクションを戻す。

`T y p e s` 指定された名前のタイプを戻す。

`I n s t a n c e s` 指定された名前のインスタンスを戻す。

`E x c e p t i o n s` 指定された名前の例外を戻す。

`A l l` 指定された名前の全てのアイテムを戻す。

`E n u m e r a t e` ネームスペース内におけるアイテムを列挙すること。

`U n m a p` このアダプタから1つのアイテムをアンマップする。

`U n m a p A l l F r o m T r e e` キャッシュから全てのアイテムをアンマップする。

`r e m o v e d` アダプタがクラスレジストリから除去されたことを当該アダプタに通告する。

#### タイプの定義

<code>V T A d a p t e r L i s t</code>	<code>V A d a p t e r N a m e S p a c e</code> オブジェクトの <code>V A r r a y</code>
<code>V T A d a p t e r R e f L i s t</code>	<code>V A d a p t e r N m a e S p a c e</code> オブジェクトの <code>V r R e f L i s t</code>
<code>V T A d a p t e r R e f</code>	<code>V A d a p t e r N m a m e S p a c e</code> オブジェクトへの <code>V r R e f e r e n c e</code>

#### `V A d a p t e r N a m e S p a c e` コンストラクタ

```

VAdapterNameSpace( VAtomRef    name,
                   VTUsageCode  usage)

```

`n a m e` ネームスペースの名前。

`u s a g e` ネームスペースのための使用フラグ。

アダプタネームスペースを作成する。このクラスは、クラスレジストリ用のオブジェクトシステムアダプタを実行するためにベースクラスとして用いられる。

`S u b S p a c e`

(113)

```

VTAdapterRef SubSpace(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1)=0

```

`name` 発見すべきネームスペースの名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

`SubSpace` は、指定された名前のネストされたネームスペースを発見する。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。

`Class`

```

VTClassRef Class(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1,
    VClassData    *last=0)=0

```

`name` 発見すべき `class` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`major` 発見すべきメジャバージョン番号。

`minor` 発見すべきマイナバージョン番号。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

`last` 最後に発見された最高バージョン番号。

`class` は、指定された名前のクラスを発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。当該アイテムのバージョンが既に発見された場合には、当該バージョンは、

(114)

バージョン番号の内部比較において最後に用いられるパラメータとしてのこの方法にパスされねばならない。

`function`

```

VFunctionRef Function(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,

    unsigned short depth=1,
    VFunctionData  *last=0)=0

```

`name` 発見すべき `function` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`major` 発見すべきメジャバージョン番号。

`minor` 発見すべきマイナバージョン番号。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

`last` 最後に発見された最高バージョン番号。

`function` は、指定された名前のファンクションを発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。当該アイテムのバージョンが既に発見された場合には、当該バージョンは、バージョン番号の内部比較において最後に用いられるパラメータとしてのこの方法にパスされねばならない。

`Type`

(115)

**VTTypeRefType(**

<b>VAtomRef</b>	<b>name,</b>
<b>VTSearchCase</b>	<b>scase=KVCaseSensitive,</b>
<b>VTVersion</b>	<b>major=KVVersionNotSpecified,</b>
<b>VTVersion</b>	<b>minor=KVVersionNotSpecified,</b>
<b>VTSearchType</b>	<b>search=KVSearchAll,</b>
<b>unsigned short</b>	<b>depth=1,</b>
<b>VTypeData</b>	<b>*last=0)=0</b>

**name** 発見すべき **type** の名前。

**scase** 探索はケース依存またはケース非依存のいずれかである。

**major** 発見すべきメジャバージョン番号。

**minor** 発見すべきマイナバージョン番号。

**search** 実施すべき探索のタイプ。

**depth** 探索すべきネームスペースの層数。

**last** 最後に発見された最高バージョン番号。

**type** は、指定された名前のタイプを発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。当該アイテムのバージョンが既に発見された場合には、当該バージョンは、バージョン番号の内部比較において最後に用いられるパラメータとしてのこの方法にパスされねばならない。

**I n s t a n c e****VTInstanceRef Instance(**

<b>VAtomRef</b>	<b>name,</b>
<b>VTSearchCase</b>	<b>scase=KVCaseSensitive,</b>
<b>VTVersion</b>	<b>major=KVVersionNotSpecified,</b>
<b>VTVersion</b>	<b>minor=KVVersionNotSpecified,</b>
<b>VTSearchType</b>	<b>search=KVSearchAll,</b>
<b>unsigned short</b>	<b>depth=1,</b>
<b>VInstanceData</b>	<b>*last=0)=0</b>

**name** 発見すべき **i n s t a n c e** の名前。

**scase** 探索はケース依存またはケース非依存のいずれかである。

(116)

`major` 発見すべきメジャバージョン番号。

`minor` 発見すべきマイナバージョン番号。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

`last` 最後に発見された最高バージョン番号。

`instance` は、指定された名前のインスタンスを発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。当該アイテムのバージョンが既に発見された場合には、当該バージョンは、バージョン番号の内部比較において最後に用いられるパラメータとしてのこの方法にパスされねばならない。

`Exception` (例外)

```

VExceptionRef Exception(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1,
    VExceptionData *last=0)=0

```

`name` 発見すべき例外の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`major` 発見すべきメジャバージョン番号。

`minor` 発見すべきマイナバージョン番号。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

`last` 最後に発見された最高バージョン番号。

`exception` は、指定された名前の例外を発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探

(117)

索される。当該アイテムのバージョンが既に発見された場合には、当該バージョンは、バージョン番号の内部比較において最後に用いられるパラメータとしてのこの方法にパスされねばならない。

## C l a s s e s

```
VTClassRefList Classes(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1)=0
```

`name` 発見すべき複数クラスの名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`major` 発見すべき最小メジャバージョン番号。

`minor` 発見すべき最小マイナバージョン番号。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

`classes` は、指定された名前の全バージョンを発見する。どちらのバージョン番号も指定されている場合には、指定された番号以上のバージョン番号を持つ全アイテムが戻される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。

## F u n c t i o n s

```
VTFunctionRefList Functions(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1)=0
```

`name` 発見すべき `functions` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

(118)

**major** 発見すべき最小メジャバージョン番号。

**minor** 発見すべき最小マイナバージョン番号。

**search** 実施すべき探索のタイプ。

**depth** 探索すべきネームスペースの層数。

**functions** は、指定された名前の全オーバロード及びバージョンを発見する。どちらのバージョン番号も指定されている場合には、指定された番号以上のバージョン番号を持つ全アイテムが戻される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。

**Types**

```

VTypeRefList Types(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1)=0

```

**name** 発見すべき **types** の名前。

**scase** 探索はケース依存またはケース非依存のいずれかである。

**major** 発見すべき最小メジャバージョン番号。

**minor** 発見すべき最小マイナバージョン番号。

**search** 実施すべき探索のタイプ。

**depth** 探索すべきネームスペースの層数。

**types** は、指定された名前の全バージョンを発見する。どちらのバージョン番号も指定されている場合には、指定された番号以上のバージョン番号を持つ全アイテムが戻される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。

**Instances**

(119)

```

VTInstanceRef fList Instances(
    VAtomRef      name,
    VTSearchCase   scase=KVCaseSensitive,
    VTVersion      major=KVVersionNotSpecified,
    VTVersion      minor=KVVersionNotSpecified,

    VTSearchType   search=KVSearchAll,
    unsigned short depth=1)=0

```

`name` 発見すべき `instances` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`major` 発見すべき最小メジャバージョン番号。

`minor` 発見すべき最小マイナバージョン番号。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

`instances` は、指定された名前のインスタンスの全バージョンを発見する。どちらのバージョン番号も指定されている場合には、指定された番号以上のバージョン番号を持つ全アイテムが戻される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。

## Exceptions

```

VTInstanceRefList Exceptions(
    VAtomRef      name,
    VTSearchCase   scase=KVCaseSensitive,
    VTVersion      major=KVVersionNotSpecified,
    VTVersion      minor=KVVersionNotSpecified,
    VTSearchType   search=KVSearchAll,
    unsigned short depth=1)=0

```

`name` 発見すべき複数例の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`major` 発見すべき最小メジャバージョン番号。

`minor` 発見すべき最小マイナバージョン番号。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

(120)

`exceptions` は、指定された名前のユーザー定義例の全バージョンを発見する。どちらのバージョン番号も指定されている場合には、指定された番号以上のバージョン番号を持つ全アイテムが戻される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。

`All`

```

VTToplevelRefList All(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,

    VTVersion      major=KVVersionNotSpecified,
    VTVersion      minor=KVVersionNotSpecified,
    VTSearchType   search=KVSearchAll,
    unsigned short depth=1,=0

```

`name` 発見すべき複数アイテムの名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`major` 発見すべき最小メジャバージョン番号。

`minor` 発見すべき最小マイナバージョン番号。

`search` 実施すべき探索のタイプ。

`depth` 探索すべきネームスペースの層数。

`all` は、指定された名前の異なる全アイテム（クラス、ファンクション、タイプ、等々）バージョンを発見する。どちらのバージョン番号も指定されている場合には、指定された番号以上のバージョン番号を持つ全アイテムが戻される。深さパラメータが1であるならば、トップレベルのネームスペースだけが探索される。

`Enumerate`

```

void Enumerate(
    VTAdapterEnumFunc func,
    void               *context,
    VTClassTag         type=KVcrAll,
    VTSearchType       search=KVsearchAll)=0

```

`func` アイテムをコールするためのファンクション。

(121)

`context` 列挙ファンクションへパスするための文脈情報。

`search` 実施すべき探索のタイプ。

`enumerate` は、指定されたタイプの全てのアイテムを列挙する。タイプが `kVcrAll` である場合には、全てのアイテムが列挙される。列挙するネストされたアイテムにおける一切のアイテムを列挙しない。発見された各アイテムに対しては、アーギュメント文脈およびアイテム自体によって列挙ファンクションがコールされる。

`Unmap`

```
void Unmap(VcrToplevel *item)=0
item      Item to unmap from the view:
```

`item` ビューからアンマップするアイテム。

`unmap` は、このアダプタのキャッシュから特定のアイテムをアンマップする。アダプタは、当該アイテムに関する全てのリファレンスを取り去り、そして、そのキャッシュから除去しなければならない。この方法は、アダプタネームスペース及び他のアイテムに適

用可能である。

`UnmapAllFromTree`

```
void UnmapAllFromTree()=0
```

`UnmapAllFromTree` は、このネームスペースにおいて再帰的に発見される全てのアイテムをこのアダプタのキャッシュからアンマップする。アダプタは、当該アイテムに関する全てのリファレンスを取り去り、そして、そのキャッシュから除去しなければならない。この方法はアダプタネームスペース及び他のアイテムに適用可能である。

`Removed`

```
void Removed()=0
```

`removed` は、アダプタが既に除去されていることを当該アダプタに通知するために、トップレベルアダプタネームスペースにおけるクラスレジストリによってコールされる。アダプタは、アイテム及びネームスペースをビューからア

ンマップしてはならない。これは、ビューが行う。removedをコールした後において、クラスレジストリは、アダプタに関するそのリファレンスを解放し、続いて当該アダプタを削除する。

### クラスVViewNameSpace

VcrTopLevelの共用サブクラス

VViewNameSpaceは、クラスレジストリアイテムのユーザによって定義されたビューを表す抽象クラスである。VAdapterNameSpacesは、名前の実階層を定義する。この場合、各オブジェクトシステムは、それ自体、他のオブジェクトシステムから分離した個別のネームスペースである。各オブジェクトシステムは、サブネームスペースを順々に含んでも差し支えない。この事前決定された名前の階層は、物理的に異なるネームスペースを見るためにエンドユーザが選定した方法を反映しないこともあり得る。例えば、ユーザーは、全てのトップレベルの名前が同一ネームスペース内に所在することを望むか、または、例えばBasicのような言語にとっては、一緒に作動するためには、平らなネームスペースを持つことが好ましいこともあり得る。これらの目標は、VViewNameSpaceを用いて達成できる。VViewNameSpaceは、名前の物理的階層と名前の階層に関するユーザーのビューとの間にユーザによって定義されるマッピングを提供する。多重VViewNameSpace階層は、物理的階層に関して存在可能である。更に、VViewNameSpaceは、アダプタネームスペースから検索されたアイテムに対して、キャッシュとして作用する。一旦、アイテムがアダプタネームスペースから検索されると、これらのアイテムは、これらのアイテムを使用するコードに影響を及ぼすことなしに、ビュー階層において動きまわることが可能である。ビューは、持続的記憶装置内に実現可能である。ビューにおいて各個別アイテムの出生場所

をセーブするために、ビューは、VcrBase::Ownerを用いて、アダプタネームスペース階層まで追跡することによりアイテムの完全に修飾された名前を発見することができる。ビューは、自身の復元に際して、完全に修飾された名前の各レベルにおいて、名前におけるその次のアイテムに関して質問すること

により、オブジェクトを作成し直すことが出来る。

共用メンバー

`VViewNameSpace` ビューネームスペースを作成する。

`DistanceTo` ツリーにおける他のビューまでの距離を戻す。

`DirectlyInView` アイテムが直接このビュー内に所在するかどうかを戻す。

`ViewOfObject` 当該オブジェクトを含むビューを戻す。

オーバーライド可能な共用メンバー

`TopLevel` この階層におけるトップレベルビューを戻す。

`SubSpace` 指定された名前のネームスペースを戻す。

`Class` 指定された名前のクラスを戻す。

`Function` 指定された名前のファンクションを戻す。

`Type` 指定された名前のタイプを戻す。

`Instance` 指定された名前のインスタンスを戻す。

`Exception` 指定された名前の例外を戻す。

`Classes` 指定された名前のクラスを戻す。

`Functions` 指定された名前のファンクションを戻す。

`Types` 指定された名前のタイプを戻す。

`Instances` 指定された名前のインスタンスを戻す。

`Exceptions` 指定された名前の例外を戻す。

`All` 指定された名前の全てのアイテムを戻す。

`Enumerate` ネームスペースにおいてアイテムを列挙する。

`MappedAdapters` このビュー上にマップされたアダプタのリストを戻す。

`Map` アイテムをこのビューにマップする。

`Unmap` このビューからアイテムをアンマップする。

`UnmapAll` このビューから全てのアダプタ特定アイテムをアンマップする。

`UnmapAllFromTree` このビュー及び含まれている全てのビュー

(124)

ーから全てのアダプタ特定アイテムをアンマップする。

**Class Changed** 当該ビュー内のクラスが変化したことをビューに  
 通告する。

**Adapter Added** 新規なアダプタがクラスレジストリにインストール  
 されたことをビューに通告する。

**Adapter Removed** クラスレジストリからアダプタが除去された  
 ことをビューに通告する。

**Removed** クラスレジストリからビューが除去されたことを当該ビュー  
 に通告する。

#### タイプの定義

<b>VTViewList</b>	<b>VViewNameSpace</b> オブジェクトの <b>VArray</b>
<b>VTViewRefList</b>	<b>VViewNameSpace</b> オブジェクトの <b>VrRefList</b>
<b>VTViewRef</b>	<b>VViewNameSpace</b> オブジェクトへの <b>VrReference</b> .

#### **VViewNameSpace** コンストラクタ

```

VViewNameSpace(   VAtomRef      name,
                   VTUsageCode  usage)

```

**name** ネームスペースの名前

**usage** ネームスペースに関する使用フラグ

ビューネームスペースを作成する。このクラスは、クラスレジストリ階層のカ  
 ストマイズ可能なビューを実行するために、ベースクラスとして用いられる。

#### **DistanceTo**

```

bool_t DistanceTo(
                   VViewNameSpace  *other,
                   int              *distance)

```

**other** ツリーにおける他のネームスペース。

**distance** (戻し) もう一方のネームスペースまでの距離。

**DistanceTo**は、同一ツリーにおける他のビューまでの距離を戻す。  
 このビューと他のビューが同一のツリーに所在しないか、又は、このビューと他

のビューが同胞である場合には、戻しはFALSEである。他のビューがこのビューの先祖であるか、又は、このビューが他のビューの先祖である場合には、TRUEが戻される。このビューが他のビューの先祖であるならば、distanceは正である（すなわち、他のビューはツリーの更に下方に位置する）。他のビューがこのビューの先祖であるならば、distanceは負である。このビューが他のビューに等しい場合には、戻しは0である。

### DirectlyInView

bool\_t DirectlyInView(VcrToplevels \*item)

item チェックするためのオブジェクト

DirectlyInViewは、アイテムがこのビュー内に直接含まれるかどうかを戻す。アイテムがこのビュー内に直接含まれない限り、方法はFALSEを戻す。

### ViewOfObject

VTVViewRef ViewOfObject(VcrToplevel \*item)

item チェックするためのオブジェクト。

ViewOfObjectは、アイテムを含むこのビュー階層におけるビューネームスペースを戻す。

### Toplevel

VViewNameSpace \*Toplevel()=0

Toplevelは、ビューネームスペースのこの階層におけるトップレベルビューを戻す。これは、しばしばコールされるので、ネストされた各ビューネームスペース内に隠されなければならない。この方法は、アイテムが特定のビュー階層内に既にマップされていたかどうかを決定するためにVcrToplevel::InViewによってコールされる。

### SubSpace

(126)

```

VTViewRef SubSpace(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1)=0

```

`name` 発見するためのネームスペースの名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索されるべきオブジェクトシステム。

`search` 遂行する探索のタイプ。

`depth` 探索されるべきネームスペースの層数。

`SubSpace` は、指定された名前のネストされたネームスペースを発見する。アイテムがキャッシュ内に発見されない場合には、ビューは、システムにマッチする各アダプタに、アイテムに関して質問しなければならない。深さのパラメータが1である場合には、トップレベルのネームスペースのみが探索される。アダプタからのアイテムが、ビュー内

において、複数の場所に現れることが可能であるかどうかを決定することはビューの責任である方法 `VcrTopLevel::InView` は、このビュー階層内に既に所在しているかどうかをチェックするために、当該アイテムに関してコールされることが可能である。

**C l a s s**

```

VTClassRef Class(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1,
    VClassData    *last=0)=0

```

`name` 発見するためのクラスの名前。

(127)

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するためのメジャバージョン番号。

`minor` 発見するためのマイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`last` 最後に発見された最高バージョン番号。

`class` は、指定された名前のクラスを発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。アイテムがキャッシュ内において発見される場合には、それよりも高いバージョン番号を持つ後続する全てのアイテムは、同一アダプタ及びアダプタネームスペースから来たかどうかだけについてチェックされる。一旦、キャッシュチェックが完了し、そして、更に高いバージョンの発見が依然として必要であるか、或いは、正確なバージョンが発見されない場合には、対応する（或いは全ての）アダプタネームスペースがチェックされなければならない。深さのパラメータが1である場合には、トップレベルネームスペースのみが検索される。アイテムのバージョンが既に発見されている場合には、そこにバージョンは、バージョン番号の内部比較に使用する最後のパラメータとして、この方法にパスされなければならない。アダプタからのアイテムが、ビュー内における複数の場所に現れることが可能であるかどうかについて決定することは、ビューの責任である。方法 `VcrTopLevel::InView` は、それが既にこのビュー階層内に所在するかどうかをチェックするために、アイテ

ムに関してコールされることが可能である。

`Function`

(128)

```

VTFunRef Function(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1,
    VFunctionData *last=0)=0

```

`name` 発見するためのファンクションの名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するためのメジャバージョン番号。

`minor` 発見するためのマイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`last` 最後に発見された最高バージョン番号。

`function`は、指定された名前のファンクションを発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。アイテムがキャッシュ内において発見される場合には、それよりも高いバージョン番号を持つ後続する全てのアイテムは、同一アダプタ及びアダプタネームスペースから来たかどうかだけについてチェックされる。一旦、キャッシュチェックが完了し、そして、更に高いバージョンの発見が依然として必要であるか、或いは、正確なバージョンが発見されない場合には、対応する（或いは全ての）アダプタネームスペースがチェックされなければならない。深さのパラメータが1である場合には、トップレベルネームスペースのみが検索される。アイテムのバージョンが既に発見されている場合には、そこにバージョンは、バージョン番号の内部比較に使用する最後のパラメータとして、この方法にパスされなければならない。アダプタからのアイテムが、ビュー内における複数の場所に現れることが可能であるかどうかについて決定することは、ビューの責任である。方法 `VcrTopLevel::InView`は、それが既にこのビュー階層内

に所在するかどうかをチェックするために、アイテムに関してコールされることが可能である。

Type

```

VTTypeRef Type(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VObjectSystem system=KVAnyObjectSystem,
    VTVersion      major=KVVersionNotSpecified,
    VTVersion      minor=KVVersionNotSpecified,
    VTSearchType   search=KVSearchAll,
    unsigned short depth=1,
    VTypeData      *last=0)=0

```

`name` 発見するためのタイプの名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するためのメジャバージョン番号。

`minor` 発見するためのマイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`last` 最後に発見された最高バージョン番号。

`type` は、指定された名前のタイプを発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。アイテムがキャッシュ内において発見される場合には、それよりも高いバージョン番号を持つ後続する全てのアイテムは、同一アダプタ及びアダプタネームスペースから来たかどうかだけについてチェックされる。一旦、キャッシュチェックが完了し、そして、更に高いバージョンの発見が依然として必要であるか、或いは、正確なバージョンが発見されない場合には、対応する（或いは全ての）アダプタネームスペースがチェックされなければならない。深さのパラメータが1である場合には、トップレベルネームスペースのみが検索される。アイテムのバージョンが既に発見されている場合には、そこにバージョンは、バージョン番号の内部比

(130)

較に使用する最後のパラメータとして、この方法にパスされなければならない。  
アダプタからのアイテムが、ビュー内における複数の場所に現れることが可能であるかどうかについて決定することは、ビューの責任である。方法 `V c r T o p L e v e l : : I n V i e w` は、それが既にこのビュー階層内に所在するかどうかをチェックするために、アイテムに関してコールされることが可能である。

`I n s t a n c e` (インスタンス)

```

VTInstanceRef Instance(
    VAtomRef          name,
    VTSearchCase     scase=KVCaseSensitive,

    VTObjectSystem   system=KVAnyObjectSystem,
    VTVersion        major=KVVersionNotSpecified,
    VTVersion        minor=KVVersionNotSpecified,
    VTSearchType     search=KVSearchAll,
    unsigned short    depth=1,
    VInstanceData    *last=0)=0

```

`n a m e` 発見するためのインスタンスの名前。

`s c a s e` 探索はケース依存またはケース非依存のいずれかである。

`s y s t e m` 探索するためのオブジェクトシステム。

`m a j o r` 発見するためのメジャバージョン番号。

`m i n o r` 発見するためのマイナバージョン番号。

`s e a r c h` 遂行するための探索のタイプ

`d e p t h` 探索するためのネームスペース層数。

`l a s t` 最後に発見された最高バージョン番号。

`i n s t a n c e` は、指定された名前のインスタンスを発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。アイテムがキャッシュ内において発見される場合には、それよりも高いバージョン番号を持つ後続する全てのアイテムは、同一アダプタ及びアダプタネームスペースから来たかどうかだけについてチェックされる。一旦、キャッシュチェックが完了し、そして、更に高いバージョンの発見が依然として必要であるか、或いは、正確なバージョンが発見されない場合には、対応する（或いは全

(131)

ての) アダプタネームスペースがチェックされなければならない。深さのパラメータが1である場合には、トップレベルネームスペースのみが検索される。アイテムのバージョンが既に発見されている場合には、そのバージョンは、バージョン番号の内部比較に使用する最後のパラメータとして、この方法にパスされなければならない。アダプタからのアイテムが、ビュー内における複数の場所に現れることが可能であるかどうかについて決定することは、ビューの責任である。方法 `VcrTopLevel::InView` は、それが既にこのビュー階層内に所在するかどうかをチェックするために、アイテムに関してコールされることが可能である。

### Exception

```

VTEExceptionRef Exception(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,

    VTSearchType  search=KVSearchAll,
    unsigned short depth=1,
    VExceptionData *last=0)=0

```

`name` 発見しようとする例外の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するためのメジャバージョン番号。

`minor` 発見するためのマイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`last` 最後に発見された最高バージョン番号。

`exception` は、指定された名前の例外を発見する。どちらのバージョン番号も指定されない場合には、最高バージョン番号を持つアイテムが選定される。アイテムがキャッシュ内において発見される場合には、それよりも高いバー

(132)

ジョン番号を持つ後続する全てのアイテムは、同一アダプタ及びアダプタネームスペースから来たかどうかだけについてチェックされる。一旦、キャッシュチェックが完了し、そして、更に高いバージョンの発見が依然として必要であるか、或いは、正確なバージョンが発見されない場合には、対応する（或いは全ての）アダプタネームスペースがチェックされなければならない。深さのパラメータが1である場合には、トップレベルネームスペースのみが検索される。アイテムのバージョンが既に発見されている場合には、そのバージョンは、バージョン番号の内部比較に使用する最後のパラメータとして、この方法にパスされなければならない。アダプタからのアイテムが、ビュー内における複数の場所に現れることが可能であるかどうかについて決定することは、ビューの責任である。方法 `V c r T o p L e v e l : : I n V i e w` は、それが既にこのビュー階層内に所在するかどうかをチェックするために、アイテムに関してコールされることが可能である。

## C l a s s e s

`VTClassRefList` `Classes`(

<code>VAtomRef</code>	<code>name,</code>
<code>VTSearchCase</code>	<code>scase=KVCaseSensitive,</code>
<code>VTObjectSystem</code>	<code>system=KVAnyObjectSystem,</code>
<code>VTVersion</code>	<code>major=KVVersionNotSpecified,</code>
<code>VTVersion</code>	<code>minor=KVVersionNotSpecified,</code>
<code>VTSearchType</code>	<code>search=KVSearchAll,</code>
<code>unsigned short</code>	<code>depth=1)=0</code>

`name` 発見しようとする `c l a s s e s` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するための最小メジャバージョン番号。

`minor` 発見するための最小マイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`C l a s s e s` は、指定された名前のクラスの全てのバージョンを発見する。

両方のバージョン番号が指定されている場合には、指定された番号よりも大きいバージョン番号を持つ全てのアイテムが戻される。一旦、ローカルビューチェックが完了すると、マップされたそれぞれのアダプタ整合システムをチェックしなければならない。深さのパラメータが1である場合には、トップレベルのネームスペースのみが探索される。アダプタからのアイテムがビュー内の複数の場所に現れることができるかどうかを決定する責任はビューにある。アイテムがこのビュー階層内に既に存在するかどうかをチェックするために、当該アイテムに関する方法 `VcrTopLevel::InView` をコールすることが可能である。

## Functions

```
VTFunctionRefList Functions(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1)=0
```

`name` 発見しようとする `functions` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するための最小メジャバージョン番号。

`minor` 発見するための最小マイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`Functions` は、指定された名前のファンクションの全てのオーバーロード及びバージョンを発見する。両方のバージョン番号が指定されている場合には、指定された番号よりも大きいバージョン番号を持つ全てのアイテムが戻される。一旦、ローカルビューチェックが完了すると、マップされたそれぞれのアダプタ整合システムをチェックしなければ

(134)

ばならない。深さのパラメータが1である場合には、トップレベルのネームスペースのみが探索される。アダプタからのアイテムがビュー内の複数の場所に現れることができるかどうかを決定する責任はビューにある。アイテムがこのビュー階層内に既に存在するかどうかをチェックするために、当該アイテムに関する方法 `VcrTopLevel::InView` をコールすることが可能である。

## Types

```
VTTypeRefList Types(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,
    VTVersion      major=KVVersionNotSpecified,
    VTVersion      minor=KVVersionNotSpecified,
    VTSearchType   search=KVSearchAll,
    unsigned short depth=1)=0
```

`name` 発見しようとする `types` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するための最小メジャバージョン番号。

`minor` 発見するための最小マイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`Types` は、指定された名前の全てのバージョンを発見する。両方のバージョン番号が指定されている場合には、指定された番号よりも大きいバージョン番号を持つ全てのアイテムが戻される。一旦、ローカルビューチェックが完了すると、マップされたそれぞれのアダプタ整合システムをチェックしなければならない。深さのパラメータが1である場合には、トップレベルのネームスペースのみが探索される。アダプタからのアイテムがビュー内の複数の場所に現れることができるかどうかを決定する責任はビューにある。アイテムがこのビュー階層内に既に存在するかどうかをチェックするために、当該アイテムに関する方法 `VcrTopLevel::InView` をコールすることが可能である。

## Instances

(135)

```

VTInstanceRefList Instances(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,

    VTVersion      major=KVVersionNotSpecified,
    VTVersion      minor=KVVersionNotSpecified,
    VTSearchType   search=KVSearchAll,
    unsigned short depth=1)=0

```

`name` 発見しようとする `instances` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するための最小メジャバージョン番号。

`minor` 発見するための最小マイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`Instances` は、指定された名前のインスタンスの全てのバージョンを発見する。両方のバージョン番号が指定されている場合には、指定された番号よりも大きいバージョン番号を持つ全てのアイテムが戻される。一旦、ローカルビューチェックが完了すると、マップされたそれぞれのアダプタ整合システムをチェックしなければならない。深さのパラメータが1である場合には、トップレベルのネームスペースのみが探索される。アダプタからのアイテムがビュー内の複数の場所に現れることができるかどうかを決定する責任はビューにある。アイテムがこのビュー階層内に既に存在するかどうかをチェックするために、当該アイテムに関する方法 `VcrTopLevel::InView` をコールすることが可能である。

`Exceptions`

(136)

```

VTEExceptionRefList Exceptions(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,
    VTVersion     major=KVVersionNotSpecified,
    VTVersion     minor=KVVersionNotSpecified,
    VTSearchType  search=KVSearchAll,
    unsigned short depth=1)=0

```

`name` 発見しようとする `exceptions` の名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するための最小メジャバージョン番号。

`minor` 発見するための最小マイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`Exceptions` は、指定された名前のユーザーによって定義された例外の全てのバージョンを発見する。両方のバージョン番号が指定されている場合には、指定された番号よりも大きいバージョン番号を持つ全てのアイテムが戻される。一旦、ローカルビューチェックが完了すると、マップされたそれぞれのアダプタ整合システムをチェックしなければならない。深さのパラメータが1である場合には、トップレベルのネームスペースのみが探索される。アダプタからのアイテムがビュー内の複数の場所に現れることができるかどうかを決定する責任はビューにある。アイテムがこのビュー階層内に既に存在するかどうかをチェックするために、当該アイテムに関する方法 `VcrTopLevel::InView` をコールすることが可能である。

A l l

(137)

```

VTToplevelRefList All(
    VAtomRef      name,
    VTSearchCase  scase=KVCaseSensitive,
    VTObjectSystem system=KVAnyObjectSystem,
    VTVersion      major=KVVersionNotSpecified,
    VTVersion      minor=KVVersionNotSpecified,
    VTSearchType   search=KVSearchAll,
    unsigned short depth=1)=0

```

`name` 発見しようとするアイテムの名前。

`scase` 探索はケース依存またはケース非依存のいずれかである。

`system` 探索するためのオブジェクトシステム。

`major` 発見するための最小メジャバージョン番号。

`minor` 発見するための最小マイナバージョン番号。

`search` 遂行するための探索のタイプ

`depth` 探索するためのネームスペース層数。

`All` は、指定された名前の全ての異なるアイテム（クラス、ファンクション、タイプ、等々）を発見する。両方のバージョン番号が指定されている場合には、指定された番号よりも大きいバージョン番号を持つ全てのアイテムが戻される。一旦、ローカルビューチェックが完了すると、マップされたそれぞれのアダプタ整合システムをチェックしなければならない。深さのパラメータが1である場合には、トップレベルのネームスペースのみが探索される。アダプタからのアイテムがビュー内の複数の場所に現れることができるかどうかを決定する責任はビューにある。アイテムがこのビュー階層内に既に存在するかどうかをチェックするために、当該アイテムに関する方法 `VcrToplevel::InView` をコールすることが可能である。

`Enumerate`

(138)

```

void Enumerate(
    VTViewEnumFunc    func,
    void               *context
    VTClassTag         type=KVcrAll,
    VTObjectSystem     system=KVAnyObjectSystem,
    VTSearchType search=KVSearchAll)=0

```

`func` 各アイテムに関してコールするためのファンクション。

`context` 列挙ファンクションへパスするための文脈情報。

`type` 探索するためのアイテムのタイプ。

`system` 探索をするためのオブジェクトシステム。

`search` 遂行するための探索のタイプ。

`Enumerate`は、指定されたタイプの全てのアイテムを列挙する。タイプが`KVcrAll`である場合には、全てのアイテムが列挙される。一旦、ローカルビューチェックが完了すれば、それぞれのマップされたアダプタ整合システムをチェックしなければならない。深さのパラメータが1である場合には、トップレベルのネームスペースのみが探索される。発見された各アイテムに関して、アーギュメント文脈およびアイテム自身と共に列挙ファンクションがコールされる。これは、ネストされた一切のネームスペースにアイテムを列挙してはならない。アダプタからのアイテムがビュー内の複数の場所に現れることが可能であるかどうかを決定するのはビューの責任である。アイテムがビュー階層内に既に存在しているかどうかをチェックするために、当該アイテムに関して、`VcrTopLevel::InView`をコールすることが出来る。

`MappedAdapters`

```

VTAdapterRefList MappedAdapters()=0

```

`MappedAdapters`は、このビューにマップされている全てのアダプタネームスペースのリストを戻す。

`Map`

```

bool_t Map(VcrToplevel *item)=0

```

`item` ビューへマップするためのアイテム。

`Map`は、特定のアイテムを現行ビューにマップする。探索および列挙に際し

(139)

てアイテムをルックアップできるように、ビューはアイテムを隠さねばならない。Mapは、クラス、ファンクション、タイプ、インスタンス、例外、及び、ネームスペースに適用される。リターンまたはTRUEは、マッピングオペレーションが成功したことを示す。リターン

が成功しても、オブジェクトは、このネームスペースにおいて直接マップ可能でないこともあり得ることに注意されたい。オブジェクトは、このビューツリー内の他の場所にマップされていることもあり得る。

Unmap

`void Unmap(VcrToplevel *item)=0`

`item` ビューからアンマップするためのアイテム。

Unmapは、現在のビューから特定のアイテムをアンマップする。アイテムがこのビューにマップされる場合には、そのアイテムは除去されなければならない。Unmapは、クラス、ファンクション、タイプ、インスタンス、例外、及び、ネームスペースに適用される。

UnmapAll

`void UnmapAll(VTObjectSystem system)=0`

`system` そのためにアイテムをアンマップするためのオブジェクトシステム。

UnmapAllは、システムによって所有される全てのアイテムを現在のビューからアンマップする。システムによって所有されるビュー内にマップされているあらゆるアイテムは除去されなければならない。UnmapAllは、クラス、ファンクション、タイプ、インスタンス、及び、例外に関するネームスペースのみに適用されない。

UnmapAllFromTree

`void UnmapAllFromTree(VTObjectSystem system)=0`

`system` そのためにアイテムをアンマップするためのオブジェクトシステム。

UnmappedAllは、現在のビュー及び含まれる全てのビューから、シ

(140)

システムによって所有される全てのアイテムを再帰的にアンマップする。ビュー内にマップされているシステム所有のあらゆるアイテムは除去されなければならない。UnmapAllFromTreeは、クラス、ファンクション、タイプ、インスタンス、及び、例外に関するネームスペースのみに適用されない。

### ClassChanged

```
void ClassChanged( VClassData      *cls,
                  short   nummeths,
                  short   numprops,
                  short   numtypes,
                  short   numinsts,
                  short   numexcepts)=0
```

cls 変化したクラス。

nummeths クラスにおける新規なmethodsの数。

numprops クラスにおける新規なpropertiesの数。

numtypes クラスにおける新規なtypesの数。

numinsts クラスにおける新規なinstancesの数。

numexcepts クラスにおける新規なexceptionsの数。

ClassChangedは、クラスを含むビューが変化したことを当該ビューに通告する。ClassChangedは、クラスに加えられた新規アイテムの数をビューに提供する。これは、専用表の内容全体に亘って複雑な探索の実施を可能にするために、ビューが専用表を管理することを可能にする。

### AdapterAdded

```
void AdapterAdded(VAdapterNameSpace *nspace)=0
```

nspace 加えられたアダプタ。

アダプタが新規にインストールされた場合には、クラスレジストリコールは、各トップレベルビューに対するAdapterAddedをコールする。ビューは、どのような選択方法であっても、そのスペース内にアダプタをマップすることが可能である。

### AdapterRemoved

```
void AdapterRemoved(VAdapterNameSpace *nspace)=0
```

(141)

n s p a c e 除去されたアダプタ。

アダプタが除去された場合、クラスレジストリは、各トップレベルビューに関するAdapterRemovedをコールする。ビューは、この時点において、アダプタのアイテムが、その階層構造内のどの位置に所在するかに関する情報を持続的に記憶することが可能である。この後において、ビューは、全てのアダプタネームスペース及び当該アダプタに属するアイテムをアンマップしなければならない。

R e m o v e d

void Removed()=0

利用可能なビューのリストからビューが除去されたことを当該ビューに通告するために、トップレベルビューに関するクラスレジストリによって、Removedがコールされる。ビューは、この時点におけるその階層に関する情報を持続的に記憶することが出来る。従って、ビューは、全てのアイテムとアダプタ、及び、全てのサブビューをアンマップしなければならない。クラスレジストリは、Removedをコールした後において、ビューを削除する原因となるべき当該ビューに関するリファレンスを解除する。

### クラスVClassRegistry

VPrimaryの共用サブクラス

VClassRegistryは、ビュー及びアダプタを管理するクラスである。アダプタは、異なるオブジェクトシステムとビューとの間のインタフェースを提供する。ビューは、ユーザーに呈示可能なネームスペースの階層を提供する。ビューは、アイテムをユーザー定義のカテゴリにグルーピングするために、ユーザーによって定義することが出来る。任意の所定クラスは、ビュー内にただ1度だけ現れることが可能であるが、同時に、多重ビューの存在が可能である。あらゆるビューは、あらゆるネームのレイアウトを持つことができる。名前（ネームスペース内のクラス、等々）の物理的階層構造は同じ状態に留どまり、そして、ビューのみが、それら自身の必要性に適するように、それらの階層構造を順序付けし直す。ビューは、ネームスペース及びアイテムの「仮想」階層を記憶する

(142)

持続的記憶オブジェクトとして実行可能であり、そして、後で階層を作り直した場合に、ロードし直す。アダプタビューは、名前の物理的階層構造を提供する。

共用メンバー

<code>VClassRegistry</code>	<code>VClassRegistry</code> オブジェクトの作成
<code>~VClassRegistry</code>	<code>VClassRegistry</code> の破壊

共用メンバー

`TypeManager` このレジストリによって使用されるタイプマネージャを戻す。

オーバーライド可能な共用メンバー

`AddView` 新規なトップレベルビューを加える。

`RemoveView` レジストリからビューを除去する。

`Views` レジストリにおける全てのビューを戻す。

`AddAdapter` オブジェクトシステムアダプタをレジストリに加える。  
。

`RemoveAdapter` レジストリからオブジェクトシステムアダプタを除去する。

`Adapters` インストールされたオブジェクトシステムアダプタのリストを戻す。

`VClassRegistry` コンストラクタ

`VClassRegistry(VTypeManager *manager)`

`manager` タイプ比較のために使用するためのタイプマネージャ。

`VClassRegistry` コンストラクタは内部データ構造である。アーギュメン

トは、タイプに関する質問のためにどのタイプマネージャをの使用するかを指定する。クラスレジストリ自体は、タイプマネージャを直接使用しない。`VClassRegistry` コンストラクタは、クラス、ファンクション、インスタンス、例外、及び、タイプ記述に含まれるタイプに関する比較情報を、クラスレジストリのユーザーが決定可能にする。

(143)

**VClassRegistry**デストラクタ**~VClassRegistry()**

**VClassRegistry**デストラクタは、全てのビューを除去し、そして、全てのアダプタを除去する。全てのクラスレジストリオブジェクトは、依然としてリファレンスされていない限り、このデストラクタによって破壊される。

**TypeManager****VTypeManager \*TypeManager()=0**

**TypeManager**は、このクラスレジストリと関連しているタイプマネージャを戻す。タイプマネージャは、データタイプの両立性を管理するために用いられる。

**AddView****void AddView(VViewNameSpace \*view)**

**view** リストに加えるためのビュー。

**AddView**は、トップレベルビューのリストに新規なビューを加える。レジストリは、ビューにリファレンスを加え、その後において、レジストリに添付された全てのアダプタに、それを通告する。

**RemoveView****void RemoveView(VViewNameSpace \*view)**

**view** リストから除去するためのビュー

**RemoveView**は、トップレベルビューのリストからビューを除去する。レジストリは、ビューに関する**Removed**をコールし、その後において、当該ビューに対するそのリファレンスを解放する。

**Views****VTViewRefList Views()**

**Views**は、クラスレジストリに添付された全ての作成されたトップレベルビューのリストを戻す。このリストは修正されてはならない。

**AddAdapter****void AddAdapter(VAdapterNameSpace \*adapter)**

(144)

`adapter` リストに加えるためのアダプタ。

`AddAdapter` は、サポートされているアダプタのリストにオブジェクトシステムアダプタを加える。各アダプタが加えられる場合には、アダプタはアダプタリストの末尾に置かれる。アダプタは同じ順序で質問されるので、アダプタがクラスレジストリに加えられる順序は重要である。これは、新規にアダプタが加えられたことを、既存の各ビューに通告する。

`RemoveAdapter`

`void RemoveAdapter(VAdapterNameSpace *adapter)`

`adapter` リストから除去するためのアダプタ。

`RemoveAdapter` は、サポートされたアダプタのリストからオブジェクトシステムアダプタを除去する。アダプタは、任意の順序において除去することが出来る。これは、アダプタが除去されたことを既存の各ビューに通告する。次に、レジストリは、当該アダプタに関する `Removed` をコールし、そして、当該アダプタに関するそのリファレンスを解除する。

`Adapters`

`VTAdapterRefList Adapters()`

`Adapters` は、インストールされた全てのオブジェクトシステムのリストを戻す。このリストは修正されてはならない。

### Implementation (具体化) クラス

これらのクラスは、露出されたクラスのセクションにおいて定義された抽象クラスのインプリメンテーションである。これらは、直接インスタンス化可能であり、そして、コードにおけるクラスレジストリエントリを作り上げるために用いることが出来る。これらのクラスは、オブジェクトシステムアダプタに対して特殊なサポートを提供するために、サブクラス化することも可能である。これらのクラスをサブクラス化するかどうか、又は、露出されたクラスのセクションにおいて定義されたクラスをサブクラス化するかどうかを決定することは、オブジェクトシステムアダプタのデザイナの裁量による。

### クラス `VcrCodedFunction`

`VFunctionData` の共用サブクラス

(145)

VFunctionDataが、アーギュメント及び例外を順々に加えることによってオブジェクトを簡潔なC++コードに作り上げることを可能にする場合には、VcrCodedFunctionはインプリメンテーションである。このインプリメンテーション

は、アーギュメント及び例外に関する情報を記憶する。このクラスは、他のオブジェクトシステムによってサブクラス化可能である。

共用メンバー

VcrCodedFunction VcrCodedFunctionオブジェクトを作成する。

AddArgument アーギュメントをファンクションに加える。

RemoveArgument ファンクションからアーギュメントを除去する。

AddException ファンクションが投げることのできる例外を加える。

RemoveException ファンクションが投げることのできる例外を除去する。

保護されたメンバー

PutType ファンクションのタイプをセットする。

保護されたデータメンバー

bool\_t itIsMethod これが方法であるかどうかを指定する。

void\*itsEntryPoint このファンクションのエントリーポイントを記憶する。

VTCallType itsCallType このファンクションのための呼出しコンベンションを記憶する。

VcrCodedFunctionコンストラクタ

(146)

```

VcrCodedFunction( VAtomRef    name,
                  VTypeData    *resultType=0,
                  bool_t       isMethod=TRUE,
                  VTUsageCode   ucodes=0,
                  void          *entryPoint=0,
                  VTCallType     calltype=KVCallTypeAnsic)

```

`name` 方法の名前。

`resultType` 方法のタイプを戻す。

`isMethod` このオブジェクトは方法を表す。

`ucodes` 方法に関する使用フラグ（該当する場合）。

`entryPoint` 方法のコール可能なエントリーポイント。

`calltype` 方法をコールする場合に使用するための呼び出しコンベンション。

ファンクション記述オブジェクトを作成する。`isMethod`が真であるならば、コ

ンストラクタは、タイプ `kVTypeObjectPointer` の隠された第1のアーギュメントを自動的に加える。作成した後において、`AddArgument` を用いて記述にアーギュメントを加えることができる。ファンクションの定義が完了した後において、その定義をクラス又はネームスペースに記憶することが出来る。

`AddArgument`

```
void AddArgument(VcrArgument *arg)
```

`arg` ファンクション記述に加えるためのアーギュメント。

`AddArgument` は、ファンクション記述にアーギュメントを加える。各アーギュメントが加えられるにつれて、アーギュメントは、アーギュメントリストの末端に配置される。クラスレジストリに直接または間接にファンクションがインストールされた後において、アーギュメントをファンクションに決して加えてはならない。

`RemoveArgument`

(147)

**void RemoveArgument(VcrArgument \*arg)**

**arg** ファンクション記述から除去するためのアーギュメント。

**RemoveArgument**は、ファンクション記述からアーギュメントを除去する。アーギュメントは、任意の順序で除去することが出来る。クラスレジストリに直接または間接にファンクションがインストールされた後において、アーギュメントをファンクションから決して除去し加えてはならない。

**AddException**

**void AddException(VExceptionData \*except)**

**except** ファンクション記述に加えるための例外。

**AddException**は、ファンクションが投げることのできる例外のリストに例外を加える。名前によって言語が例外にアクセスすることを可能にするためには、クラスまたはネームスペースのいずれかにも例外を加えなければならない。

**RemoveException**

**void RemoveException(VExceptionData \*except)**

**except** ファンクション記述から除去するための例外。

**RemoveException**は、ファンクションが投げることのできる例外のリストから例外を除去する。例外は、任意の順序において除去することができる。ファンクションがクラスレジストリに直接または間接にインストールされた後においては、ファンクションから例外を決して除去してはならない。

**PutType**

**void PutType(VTypeData \*type)**

**type** ファンクションのタイプ。

ファンクションの戻しのタイプをセットする。これは、ベースクラスの初期化の後においてファンクションのタイプをセットする必要がある **VcrCodedFunction** のサブクラスからコールされることを意図したものである。ファンクションのタイプは、それが最初に検索された後においては、変更されてはならない。

クラスVcrCodedProp

VPropDataの共用サブクラス

VcrCodedPropは、直接C++から使用できるVPropDataのインプレメンテーションであり、従って、VPropDataのサブクラスは定義する必要がない。VcrCodedPropは、それ自身の中にアクセッサ方法を格納する。このクラスは、サブクラス化可能であるように設計される。

共用メンバー

VcrCodedProp VPropDataオブジェクトの生成

保護されたメンバー

PutType ファンクションのタイプをセットする。

保護されたメンバー

PutGetMethod 特性の「ゲット」方法をセットする。

PutSetMethod 特性の「セット」方法をセットする。

VcrCodedPropコンストラクタ

```
VcrCodedProp(VAtomRef          name,
              VTypeData   *resultType=0,
              VTUsageCode   ucodes=0,
              VFunctionData *getMethod=0,
              VFunctionData *setMethod=0)
```

name 特性の名前。

resultType 特性のタイプ。

ucodes 特性に関する使用フラグ（該当する場合）。

getMethod 「ゲット」方法の記述。

setMethod 「セット」方法の記述。

特性記述を作成する。アクセッサ方法の名前は、「ゲット」または「セット」いずれか

によって先行される特性の名前でなければならない。このネーミングコンベンションは、多重クラスレジストリによって作動化可能にされたプログラム言語を堅実に横断して特性にアクセスすることを可能にする。

## PutType

```
void PutType(VTypeData *type)
```

type 特性のタイプ。

特性のタイプをセットする。これは、ベースクラスの初期化の後において特性のタイプをセットする必要のあるVcrCodedPropのサブクラスからコールされることを意図したものである。特性のタイプは、最初に検索された後で、変更されてはならない。

## PutGetMethod

```
void PutGetMethod(VFunctionData *getMethod)
```

getMethod 「ゲット」方法の記述。

特性の「ゲット」方法を変える。通常これは、作成した後でデータをセットする必要があるサブクラスからのみコールされなければならない。

## PutSetMethod

```
void PutSetMethod(VFunctionData *setMethod)
```

setMethod 「セット」方法の記述。

特性の「セット」方法を変える。通常これは、作成した後でデータをセットする必要があるサブクラスからのみコールされなければならない。

## クラスVcrCodedClass

VClassDataの共用サブクラス

VcrCodedClassは、VClassDataをサブクラス化する必要なしにC++コードから直接クラスを定義するために使われるVClassDataのインプリメンテーションである。インスタンスは、方法、特性、タイプ、インスタンス、及び、例外のリストが加えられるにつれて、これらを記憶する。このクラスは、必要に応じて、サブクラス化されるように、設計される。

共用メンバー

VcrCodedClass

Construct a VcrCodedClass オブジェクトの作成

共用メンバー

AddMethod 方法をクラス記述に加える。

(150)

`RemoveMethod` クラス記述から方法を除去する。

`AddProperty` 特性をクラス記述に加える。

`RemoveProperty` クラス記述から特性を除去する。

`AddType` タイプをクラス記述に加える。

`RemoveType` クラス記述からタイプを除去する。

`AddInstance` インスタンスをクラス記述に加える。

`RemoveInstance` クラス記述からインスタンスを除去する。

`AddException` 例外をクラス記述に加える。

`RemoveException` クラス記述から例外を除去する。

保護されたメンバー

`PutBaseClass` このクラスのベースクラスをセットする。

`PutBaseClasses` このクラスのベースクラスのリストをセットする。

`PutConstructor` コンストラクタファンクションをセットする。  
。

`PutDuplicator` コピーコンストラクタファンクションをセットする。

`PutDestructor` デストラクタ方法をセットする。

`PutAcquireMethod` 獲得方法をセットする。

`PutReleaseMethod` リリース方法をセットする。

`MyMethods` このクラスにおいて定義される方法のリストを戻す。

`MyProperties` このクラスにおいて定義される特性のリストを戻す。

`MyTypes` このクラスにおいて定義されるタイプのリストを戻す。

`MyInstances` このクラスにおいて定義されたインスタンスのリストを戻す。

`MyExceptions` このクラスにおいて定義される例外のリストを戻す。

保護されたデータメンバー

(151)

VTVersion itsMajor クラスのメジャバージョン番号。

VTVersion itsMinor クラスのマイナバージョン番号。

VTCastList itsCastToBasesList 所定のベースクラスにキャストされるファンクションのリスト。

VTSafeCastList itsCastFromBasesList 所定のベースクラスからキャストされたファンクションのリスト。

VcrCodedClass コンストラクタ

```

VcrCodedClass(
    VAtomRef      name,
    VClassData    *baseClass=0,
    VFunctionData *constructor=0,
    VFunctionData *duplicator=0,
    VFunctionData *destructor=0,
    VFunctionData *acquire=0,
    VFunctionData *release=0,
    VTUsageCode   ucodes=0,
    VTVersion      major=0,
    VTVersion      minor=0)
VcrCodedClass(
    VAtomRef      name,
    VTClassList   baseClasses,
    VTCastList     castToBases,
    VTSafeCastList castFromBases,
    VFunctionData *constructor=0,
    VFunctionData *duplicator=0,
    VFunctionData *destructor=0,
    VFunctionData *acquire=0,
    VFunctionData *release=0,
    VTUsageCode   ucodes=0,
    VTVersion      major=0,
    VTVersion      minor=0)

```

name クラスの名前。

baseClass 単独継承されたこのクラスに関するベースクラス（或いは、NULL）。

baseClasses 多重継承されたこのクラスに関するベースクラスのリスト。

`castToBases` 各ベースクラスにキャストするために用いられるファンクションのリスト。

`castFromBases` 各ベースクラスからキャストするために使われるファンクションリスト。

`constructor` クラスコンストラクタの記述。

`duplicator` クラスコピーコンストラクタの記述。

`destructor` クラスデストラクタ（方法でなければならない）の記述。

`acquire` インスタンス（方法でなければならない）にリファレンスを加えるために用いられる方法の記述

`release` クラス（方法でなければならない）のインスタンスに対するリファレンスを除去するために用いられる方法の記述。

`usage` クラスに関する使用コード。

`major` クラスのメジャバージョン番号。

`minor` クラスのマイナバージョン番号。

クラスの記述を作成する。クラスは、単独継承されるか、又は、多重継承されたベースクラスであっても差し支えない。ベースクラスを作成するためには、コンストラクタの第1オーバロードは、ベースクラスとしてのNULLと共に使用しなければならない。

`constructor`（コンストラクタ）は、クラスのための第1のコンストラクタである。`constructor`（コンストラクタ）は、例えば、オブジェクトのインスタンスを作成するためにアプリケーションビルダによって使用されても差し支えない。言語インタプリタに関して、コンストラクタが指定されない場合には、インスタンスは作成されることは出来ないが、何らかの他の方法によって導入されなければならない。例えば、インスタンスは、ファンクションによって戻されても差し支えない。

これらの方法（コンストラクタ、コピー、デストラクタ、アクアイア、リリース）は、オブジェクト自体によって、必ずしも実行される必要がないが、クラス

レジストリに容易に統合できるように設計された利便なファンクションであっても差し支えないことに注意されたい。 `constructor`（コンストラクタ）、及び、`copy`（コピー）は任意のアーギュメントを持っても差し支えないが、デストラクタ、アクアア、及び、リリースは一切のアーギュメントを持つてはならず、或いは、全てのアーギュメントはデフォルト値を持たねばならない。

`castToBases` 及び `castFromBases` は、ゼロ長さであるか、或いは、ベースクラスと同数のエントリ数を持たなければならない。これは、ファンクションポインタのリストである。`castToBases` における各ファンクションは、アーギュメントとしてオブジェクトインスタンスをとり、そして、当該ベースクラスにキャストされる場合に、インスタンスに対して、対応するポインタを戻す。`castFromBases` における各ファンクションは、アーギュメントとしてオブジェクトインスタンス及びブーリアンをとり（キャストがタイプセーフキャストであるかどうかを指定する）、そして、当該ベースクラスからこのクラスへキャストされた場合に、インスタンスに関して、対応するポインタを戻す。任意のエントリが0であれば、ベースクラスに対してキャストされた場合に、オブジェクトが同じポインタを保持するものと仮定される。このリストは、`CastToDirectBase` 及び `CastFromDirectBase` によって使用される。C++多重継承クラスが使用される場合には、このリストが特に必要とされる。`VcrCodedClass` が、これらのキャストをする必要がないオブジェクトシステムによってサブクラス化された場合には、空のリストがコンストラクタにパスされることもあり得る。

#### `AddMethod`

```
void AddMethod(VFunctionData *method)
```

`method` クラス記述に加える方法。

`AddMethod` は方法をクラス記述に加える。`methods` は通常の方法に関して加えることが可能であり、他の `VFunctionData` オブジェクトは、クラスの静的な方法の概念をサポートするために加えることが可能である。`methods` は、クラスレジストリにおいてクラスがインストールされる

か、或いは、引用された後において加えても差し支えない。

## `RemoveMethod`

```
void RemoveMethod(VFunctionData *method)
```

`method` クラス記述から除去する方法。

`RemoveMethod`はクラス記述から方法を除去する。`Methods`は、クラス記述がクラスレジストリに直接または間接にインストールされた後において、決してクラス記述から除去してはならない。このクラス記述のスーパークラスにインストールされた方法を除去しようと試みることによって、所要の効果が得られないことに注意されたい。方法は、それが定義されたクラスから除去されなければならない。サブクラスから方法を除去することの効果は、使用フラグ `KVUsageHidden` を用いて、同じ方法記述をサブクラスのクラス記述に加えることによって生成することが出来る。

## `AddProperty`

```
void AddProperty(VPropData *prop)
```

`prop` クラス記述に加えるための特性。

`AddProperty`は特性をクラス記述に加える。特性は、クラスがクラスレジストリにインストールされるか、又は、引用された後において、加えても差し支えない。

## `RemoveProperty`

```
void RemoveProperty(VPropData *prop)
```

`prop` クラス記述から除去するための特性。

`RemoveProperty`はクラス記述から特性を除去する。特性は、クラス記述がクラスレジストリに直接又は間接にインストールされた後において、決してクラス記述から除去されてはならない。このクラス記述スーパークラスにインストールされた特性を除去しようと試みることは所要の効果を生成しないことに注意されたい。特性は、それが定義されたクラスから除去されなければならない。サブクラスから特性を除去することの効果は、使用フラグ `KVUsageHidden` を用いて同じ特性記述をサブクラスのクラス記述に加えることによって生成される。

## AddType

```
void AddType(VTypeData *type)
```

`type` クラス記述に加えるためのタイプ。

`AddType`はクラス記述にタイプを加える。`Types`は、クラスがクラスレジストリにインストールされるか、或いは、引用された後において、加えられても差し支えない。言語においては、このタイプは、この特定クラスのオブジェクトの文脈においてのみ適用可能であることに注意されたい。その代りに、一般的な使用に関してタイプを定義するためには、当該タイプをネームスペースに加えること。

## RemoveType

```
void RemoveType(VTypeData *type)
```

`type` クラス記述から除去するためのタイプ。

`RemoveType`はクラス記述からタイプを除去する。`Types`は、クラス記述がクラスレジストリに直接または間接にインストールされた後において、決してクラス記述から除去されてはならない。

## AddInstance

```
void AddInstance(VInstanceData *inst)
```

`inst` クラス記述に加えるためのインスタンス。

`AddInstance`は、名前付きインスタンスをクラス記述に加える。これは、一般に、名前付き定数に関してのみ使用されることに注意されたい。インスタンスは、クラスがクラスレジストリにおいてインストールされるか、或いは、引用された後において加えられても差し支えない。言語において、このインスタンスは、この特定のクラスのオブジェクトの文脈においてのみ適用可能であることに注意されたい。一般的な使用に関してインスタンスを定義するためには、ネームスペースに当該インスタンスを加えること。

## RemoveInstance

```
void RemoveInstance(VInstanceData *inst)
```

`inst` クラス記述から除去するためのインスタンス。

`RemoveInstance`はクラス記述から名前付きインスタンスを除去する。インスタンスは、クラス記述がクラスレジストリに直接または間接にインストールされた後において、決してクラス記述から除去されてはならない。

#### `AddException`

`void AddException(VExceptionData *except)`

`except` クラス記述に加えるための例外。

`AddException`は例外をクラス記述に加える。例外は、クラスレジストリに

においてクラスがインストールされるか又は引用された後において、加えられても差し支えない。言語において、この例外は、この特定のクラスのオブジェクトの文脈においてのみ適用可能であることに注意されたい。一般的な使用に関して例外を定義するためには、その代わりに、当該例外をネームスペースに加えること。

#### `RemoveException`

`void RemoveException(VExceptionData *except)`

`except` クラス記述から除去するための例外。

`RemoveException`はクラス記述から例外を除去する。例外は、クラスレジストリクラスにおいてクラス記述が直接または間接にインストールされた後において、決してクラス記述から除去されてはならない。

#### `PutBaseClass`

`void PutBaseClass(VTClassData *base)`

`base` このクラスのベースクラス。

`PutBaseClass`は、記述されたクラスに関してベースクラスをセットする。クラスが多重的に継承された場合には、その代りに、`PutBaseClasses`を使用しなければならない。この方法は、動的に生成されたベースクラスを記憶するために、サブクラスによって使用することが出来る。

#### `PutBaseClasses`

`void PutBaseClass(VTClassList bases)`

(157)

`bases` このクラスのベースクラス。

`PutBaseClasses` は記述されたクラスに関するベースクラスのリストをセットする。この方法は、動的に生成されたベースクラスを記憶するために、サブクラスによって使用することが出来る。

`PutConstructor`

`void PutConstructor(VFunctionData *constructor)`

`constructor` クラスに関するコンストラクタ。

`PutConstructor` は、記述されたクラスに関するコンストラクタをセットする。このコンストラクタは、オブジェクトに関するC++コンストラクタと同じでない。クラスレジストリによって定義されたコンストラクタは、オブジェクトの割当てに関して責任があり、当該コンストラクタは新規なオブジェクトを戻さなければならない。コンストラクタに関するクラスレジストリ記述は総称オブジェクトリターンタイプ（タイプコード `KVTypeObjectPointer`）を持つ。この方法を用いて格納された `VF`

`unctionData` は、オブジェクトインスタンスに関する隠された第1のアーギュメントを持たない。従って、この `VFunctionData` は方法とみなしてはならない。

`PutDuplicator`

`void PutDuplicator(VFunctionData *duplicator)`

`duplicator` クラスに関するコピーコンストラクタである。

`PutDuplicator` は、記述されたクラスに関するコピーコンストラクタをセットする。このコンストラクタは、オブジェクトに関するC++コピーコンストラクタと同じではない。クラスレジストリによって定義されたコピーコンストラクタはオブジェクトの割当てに関して責任があり、そして、新規なコピーデュプリケートオブジェクトを戻さねばならない。コピーコンストラクタに関するクラスレジストリ記述は総称オブジェクトリターンタイプ（タイプコード `KVTypeObjectPointer`）を持つ。この方法を用いて格納された `VFunctionData` は、コピーされつつあるオブジェクトインスタンス

に関する隠された第1のアーギュメントを持つ。従って、この `VFunctionData` は方法であるとみなすことが出来る。

`PutDestructor`

`void PutDestructor(VFunctionData *destructor)`

`destructor` クラスのためのコンストラクタ

`PutDestructor` は記述されたクラスに関するデストラクタをセットする。この方法は、一切のアーギュメントを持たないか、或いは、全てのアーギュメントに関してデフォルト値を持たなければならない。このデストラクタ方法は、オブジェクトインスタンスに対して割当てられた記憶装置の実自由の実施に関して責任があると言う点において、C++デストラクタと同じではない。`destructor` は方法でなくてはならない。

`PutAcquireMethod`

`void PutAcquireMethod(VFunction *acquire)`

`acquire` クラスに関するアクワイア（獲得）方法。

`PutAcquireMethod` は、クラスのインスタンスにリファレンスを加えるために用いられる方法をセットする。この方法は、一切のアーギュメントを持ってはならないか、或いは、全てのアーギュメントがデフォルト値を持たなければならない。`acquire` は方法でなければならない。オブジェクトシステムがリファレンスカウントを継承的にサポートしない間合いには、クラスレジストリは、`classes` ユーティリティを用いて、インスタンスをベースとするリファレンスカウントに関してサポートを提供する。

ることができる。

`PutReleaseMethod`

`void PutReleaseMethod(VFunctionData *release)`

`release` クラスのための解除（リリース）方法。

`PutReleaseMethod` は、クラスのインスタンスに関するリファレンスを除去するために用いられる方法をセットする。この方法は、一切のアーギュメントを持ってはならず、或いは、全てのアーギュメントがデフォルト値を

持たなければならない。releaseは方法でなければならない。オブジェクトシステムがリファレンスカウントを継承的にサポートしない場合には、クラスレジストリは、classesユーティリティを用いて、インスタンスをベースとするリファレンスカウントをサポートすることができる。

### MyMethods

#### VTFunctionList MyMethods()

MyMethodsは、記述されたこのクラスにおいて定義された方法のリストを返す。これは、AddMethodを用いて加えられた方法のリストのみを返す。これは、記述されたクラスのスーパークラスの方法は一切戻さない。これは、一般に、どの方法記述がこのクラス記述に関して既に動的に生成済みであるかをVcrCodedClassのサブクラスが決定することを支援するために用いられる。

### MyProperties

#### VTPropList MyProperties()

MyPropertiesは、記述されたこのクラスにおいて定義された特性のリストを返す。これは、AddPropertyを用いて加えられた特性のリストのみを返す。これは、記述されたこのクラスのスーパークラスの特性は一切戻さない。これは、一般に、どの特性記述がこのクラス記述に関して既に動的に生成済みであるかをVcrCodedClassのサブクラスが決定することを支援するために用いられる。

### MyTypes

#### VTTypeList MyTypes()

MyTypeは、記述されたこのクラスにおいて定義されたタイプのリストを返す。これは、AddTypeを用いて加えられたタイプのリストのみを返す。これは、記述されたこのクラスのスーパークラスのタイプは一切戻さない。これは、一般に、どの方法記述がこのクラス記述に関して既に動的に生成済みであるかをVcrCodedClassのサブクラスが決定することを支援するために用いられる。

## MyInstances

VTInstanceList MyInstance()

MyInstancesは、記述されたこのクラスにおいて定義されたインスタンスのリストを返す。これは、AddInstanceを用いて加えられたインスタンスのリストのみを返す。これは、記述されたこのクラスのスーパークラスのインスタンスは一切戻さない。これは、一般に、どのインスタンス記述がこのクラス記述に関して既に動的に生成済みであるかをVcrCodedClassのサブクラスが決定することを支援するために用いられる。

## MyExceptions

VTExceptionList MyExceptions()

MyExceptionsは、記述されたこのクラスにおいて定義された例外のリストのみを返す。これは、AddExceptionを用いて加えられた例外のリストのみを返す。これは、記述されたこのクラスのスーパークラスの例外は一切戻さない。これは、一般に、どのインスタンス記述がこの例外記述に関して既に動的に生成済みであるかをVcrCodedClassのサブクラスが決定することを支援するために用いられる。

## クラスVcrSimpleAdapter

VAdapterNameSpaceの共用サブクラス

VcrSimpleAdapterは、オブジェクトシステムアダプタに関するベースインプリメンテーションとして使用できるVAdapterNameSpaceのインプリメンテーションである。これは、VAdapterNameSpaceの全ての方法のサポートを提供する。オブジェクトシステムは、これをサブクラス化し、VcrSimpleAdapterによって提供されたキャッシングサポートを用いて、異なる方法における動的ルックアップを提供することができる。異なるオブジェクトシステムに関するネームスペースを作成する際にVAdapterNameSpace又はVcrSimpleAdapterのいずれかサブクラス化が可能である。

これは、そのInstall、Remove方法、または、付加的機能を提供するためにオブジェクトシステムアダプタによってサブクラス化することのでき

(161)

る方法を介して直接使用することのできる簡単なアダプタ構造を提供する。

オーバーライド可能な共用メンバー

`Install` アダプタにアイテムをインストールする。

`Remove` アダプタからアイテムを除去する。

保護されたオーバーライド可能なメンバー

`SubSpaces` この中のサブスペースのリストを戻す。

`Install`

```
void Install(VcrToplevels *item)
```

`item` ネームスペースにインストールするためのアイテム。

`Install`は、このコードネームスペース内に特定のアイテムをインストールする。`Install`は、クラス、ファンクション、タイプ、インスタンス、及び、例外のみに適用される。オブジェクトが一旦作成されると、これは、コードアダプタにインストール可能であり、従って、このオブジェクトは、異なるビューにとって可視である。

`Remove`

```
void Remove(VcrToplevel *item)
```

`item` ネームスペースから除去するためのアイテム。

`Remove`は、ネームスペースから特定のアイテムを除去する。アイテムが任意のビュー内にマップされている場合には、当該アイテムは全てのビューからアンマップされる。`Remove`は、コードネームスペース、クラス、ファンクション、タイプ、インスタンス、及び、例外に適用される。

`SubSpaces`

```
VTAdapterList SubSpaces(VTSearchType stype)
```

`stype` 実施される探索のタイプ。

`SubSpaces`は、このネームスペース内に含まれるアダプタネームスペースのリストを検索する。この方法は、再帰的探索（即ち、深さが1以外の探索）に実施を支援するために他の方法のベースクラスインプレメンテーションによってコールされる。サブクラスは、ネストにされたネームスペースの動的ルック

アップに関するサポートを提供するかに、この方法を再実行することが出来る。

### クラスVcrCodeAdapter

VcrSimpleAdapterの共用サブクラス

VcrCodeAdapterは、それ自身の中のキャッシュテーブルにおけるC++コードにおいて直接作られたアイテムを格納するVcrSimpleAdapterのインプレメンテーションである。これは、そのInstall、Remove、及び、CreateSubSpace方法を介して直接使用することのできる簡単なアダプタ構造を提供する。VcrSimpleAdapterによって提供される機能性の最上位において、このクラスは、ネストされたネームスペースのエンドユーザーによる作成に関するサポートを提供する。アダプタのユーザーは、一般に、アダプタ内にネストされたネームス

ペースを（露出mixinからのサポートなしに）作ることが許容されないので、オブジェクトシステムアダプタは、その代りに、VcrSimpleAdapterをサブクラス化しなければならない。

共用メンバー

CreateSubSpace その中にネストされたネームスペースを作成する。

CreateSubSpace

VTAdapterRef CreateSubSpace(VAtomRef name)

name 新規なネームスペースを作るための名前。

CreateSubSpaceは、新規なアダプタネームスペースを作り、そして、それを、現行ネームスペース内にインストールする。その代りに、指定された名前のネームスペースが既に存在する場合には、それを戻さねばならない。

### クラスVcrSimpleView

VViewNameSpaceの共用サブクラス

VcrSimpleViewは、アイテムをアダプタから隠すために必要な機能性のベースレベルを提供するVViewNameSpaceのインプレメンテ

ーションである。これは、アダプタネームスペースをそれ自身内にマッピングするための簡単なポリシーを提供する。各アダプタは、トップレベルネームスペースを共有する。アダプタの中のトップレベルネームスペースが発見されるにつれて、これらのスペースはトップレベルネームスペース上にマップされる。マップされたこれらのネームスペース内における階層は、アダプタ自体において定義された階層構造を表す。持続的な組み込みサポートは存在しない。このクラスは、異なるマッピングポリシーを提供するための持続的なサポートを提供するためにサブクラス化しても差し支えない。マッピングポリシーは、`AdapterAdded`及び`MapAdapter`をオーバーライドすることによって変更可能である。持続的サポートは、`Removed`及び`AdapterRemoved`をオーバーライドすることによって提供可能である。

保護されたメンバー

`Install` ビュー内にアイテムをインストールする。

`Remove` ビューからアイテムを除去する。

`CanMap` アダプタアイテムがマップ可能であるかどうかを戻す。

保護されたオーバーライド可能なメンバー

`CreateSubSpace` その中にネストされたネームスペースを作る

。

`SubSpaces` その中におけるサブスペースのリストを戻す。

`Install`

`void install(VcrToplevel *item)`

`item` ネームスペース内にインストールするためのアイテム。

`Install`は、このビューネームスペース内に特定のアイテムをインストールする。`Install`は、クラス、ファンクション、タイプ、インスタンス、例外、アダプタ、及び、ビューに適用される。オブジェクトがアダプタから一旦検索されると、それは、ビュー内にインストール可能であり、従って、クラスレジストリのユーザーにはそれが見える。`Install`は、アイテムをビュー内に直接インストールする。この方法は、全ての必要なリファレンスカウントオ

ペレーションを行う。

## Remove

```
void Remove(VcrToplevel *item)
```

**Item** ネームスペースから除去するためのアイテム

**Remove**は、特定のアイテムをネームスペースから除去する。**Remove**は、ビュー、アダプタ、クラス、ファンクション、インスタンス、及び、例外に適用される。この方法は、全ての必要なリファレンスカウントオペレーションを行う。

## CanMap

```
void Remove(VcrToplevel *item)
```

**item** チェックするためのアイテム。

**CanMap**は、アイテムがこのビューにマップ可能であるかどうかを返す。この方法のリターンがFALSEである場合には、他の一切のビュー方法はアイテムを戻してはならない。これは、アイテムがビュー方法から戻されることが可能かどうかをチェックするために、内部的にコールされる。デフォルトインプレメンテーションは、**item**→**InView**（これ）をコールした結果を返す。この方法は、ビュー内の複数の場所へアイテムをマップすることのサポートを提供するために、サブクラスにおいて再実行可能である。

## CreateSubSpace

```
VTViewRef CreateSubSpace(VAtomRef name)
```

**name** 新規なネームスペースを作るための名前

**CreateSubSpace**は、新規なビューネームスペースを作り、そして、現在のネームスペース内にそれをインストールする。指定された名前のネームスペースが既に存在する場合には、その代わりに、それが戻されなければならない。ネストされたネームスペースの正しいクラスを作ることが出来るように、この方法は、サブクラスにおいてオ

ーバライドされるなければならない。

## SubSpaces

### VTViewList SubSpaces(VTSearchType stype)

stype 遂行するための探索のタイプ。

SubSpacesは、このネームスペースの中に含まれるビューネームスペースのリストを戻す。この方法は、持続的に格納されたビュー情報を検索するためのサポートを提供するために、サブクラスにおいてオーバーライド可能である。

### クラスVcrFlatView

VcrSimpleViewの共用サブクラス

VcrFlatViewは、フラットネームスペースモデルを提供するVViewNamespaceのインプレメンテーションである。各アダプタネームスペースが発見されるにつれて、これは、ビューのトップレベルネームスペース上にマップされる。全てのアイテムは、当該ビューのトップレベルネームスペースにおいて発見可能である。このクラスは、多重ネームスペースの概念を処理する方法を持たない例えばBasicのような言語によって使用可能である。このクラスは、サブクラス化されてはならない。フラットネームスペースの異なるインプレメンテーションを提供するために、VcrSimpleViewは、その代りにサブクラス化されなければならない。

### Type Management (タイプ管理) クラス

複合タイプの定義

可能なタイプの集合が基本タイプに制限された場合において、クラスレジストリは、全ての定義されたクラスの小さい部分集合を記述するためにのみ有用である。ただし、可能なタイプの集合は静的ではない。あらゆる複合的なデータタイプを記述するために新規なタイプを作ることが出来る。

基本タイプ、及び、例えば、構造、結合、及び、enumsのような複合タイプは、VTypeDataのサブクラスを用いて記述することが出来る。タイプの定義は、複合的なデータタイプを任意に作るためにネストすることが出来る。タイプ管理システムは、全ての可能なC++タイプをタイプ記述にマップすることを意図したものではない。これは、あらゆる標準オブジェクトシステムタイプを記述することができることを意図する。タイプ管理システムが記述することのできるタイプの集合は、CORBA IDL、及び、CORBA TypeCo

d e s 内に記述することのできる全てのタイプを含む。記述可能なタイプの集合はオブジェクトを含まない。オブジェクト記述は、他のクラスレジストリ V C l a s s D a t a インスタンスによって行われる。

タイプは、タイプ記述オブジェクトの階層を組み立てることによって記述される。タイ

プが一旦作られると、クラスレジストリにおいて直接使用することが出来る。タイプクラスは、1つのタイプのインスタンスを他のタイプのインスタンスにキャストすることを容易にする。構造が類似したタイプは相互にキャスト可能である。C 及び C ++ とは異なり、クラスレジストリタイプシステムはフィールドのデータタイプが異なるとしても、フィールド記述が相互に類似した構造をキャストする。この相関性は、オブジェクトシステムにブリッジング能力を提供するために用いられる。

サポートコード (V c r C a l l、及び、V c r C o m p l e t e C a l l) をコールするクラスレジストリファンクション及び方法は、アーギュメントをキャストし、そして、値を該当するタイプに戻すためにタイプ管理システムを用いる。サポートコードをコールするファンクションは、タイプアーギュメント及びリターンタイプが正しくキャストされることを確認するためにタイプチェックを行うが、これらのファンクションをコールする以前に厳密にタイプチェックすることはユーザーの責任である。

各タイプオブジェクトは、それ自体をタイプコードと関連付けることが出来る。これらのタイプコードは、クラスレジストリの実行中のインスタンス内のタイプを一意的に記述する。タイプコードは、迅速なタイプ比較のために使用できる。タイプマネージャーは、タイプコードの管理に責任がある。タイプマネージャーは、そのデータベースに同じタイプ記述が無い場合には、1つのタイプに対して一意的なタイプコードを生成する。既に同じタイプが存在する場合には、双方のタイプに同じタイプコードが与えられる。従って、同一のタイプコードを持つインスタンスは同じである。これは、タイプコードの異なる2つのインスタンスはレイアウト及び構造が同じでないことを意味しない。異なる複合タイプは、フ

フィールド及びサブタイプのレイアウトが同じであっても差し支えないが、フィールド名が異なる場合には、2つのタイプには異なるタイプコードが与えられる。

事前定義済みのタイプを除いては、タイプコードは持続的でない。全ての異なるプロセススペースは、記述されたタイプに関して異なるタイプコードを持つ。ただし、タイプコードはタイプ比較においてのみ使用されるので、これは、通常、問題にはならない。一組の持続的な利用可能なタイプコードがある。これらは、必要に応じて、Visual Edgeによって割り当てられる。持続的なタイプコードは、基本的なタイプ記述に限って割り当てられる。タイプマネージャーは、全ての持続的なタイプコードを追跡することに責任がある。持続的なタイプは、その記述オブジェクトを発見するために、必ず、タイプマネージャーにおいてルックアップ可能である。

タイプがオブジェクトによって記述されるので、これらのオブジェクトはmixinを持っても差し支えない。タイプに関するmixinは、タイプに関する付加的情報を開発環境に提供するために用いられる。たとえば、「カラー」と命名されたタイプは、特殊化されたカラーエディターを展開するためのmixinサポートを持つと言う点を除けば符号無しロングと同じであっても差し支えない。

#### 事前定義済みタイプ (Predefined Types)

事前定義済みタイプの集合は、C++基本タイプ、並びに、例えばストリング及び総称オブジェクトポインタのような多数の簡単な合成タイプの全範囲に亘ってカバーする。

#### 基本タイプ (Fundamental Types)

基本タイプ及び他の事前定義済みタイプの集合は、タイプマネージャーに記憶された記述を持つ。これらのタイプ記述は、タイプコードを用いて、タイプマネージャーに質問することによって発見できる。現在定義されているタイプコードの集合を次に示す：

(168)

KVTypeVoid	KVTypeByte	
KVTypeSignedChar	KVTypeUnsignedChar	KVTypeChar
KVTypeSignedShort	KVTypeUnsignedShort	KVTypeShort
KVTypeSignedInt	KVTypeUnsignedInt	KVTypeInt
KVTypeSignedLong	KVTypeUnsignedLong	KVTypeLong
KVTypeFloat	KVTypeDouble	KVTypeLongDouble

### 他の事前定義済みタイプ (Other Predefined Types)

KVTypeVoidPointer  
KVTypeObjectPointer  
KVTypeFunctionPointer

次に示すタイプは、それらのタイプコードが複数のセッションを通じて持続的であるという点で、他の記述済み複合タイプと異なる。

KVTypeSignedString	サイン付ストリング
KVTypeUnsignedString	サイン無しストリング
KVTypeString	ネイティブプラットフォームのためのデフォルト サインのストリング
KVTypeBString	基本言語ストリング
KVTypeVariant	OLE 2.0 の VARIANT

### クラス VTypeData

VcrTopLevel の共用サブクラス

VTypeData は、全てのタイプ記述クラスに関するベースクラスである。これは、タイプ比較、キャスト、作成、及び、破壊に関して仮想方法を提供する。

保護されたメンバー

PutTypeManager このインスタンスを所有するタイプマネージャーをセットする。

PutBaseType これがベースとするタイプをセットする。

GetActualInstance タイプのインスタンスに関する実タイプおよびポインタを入手すること。

共用メンバー (Public Members)

(169)

`BaseType` このタイプのベースタイプを戻す。

`TypeManagcr` このオブジェクトと関連したタイプマネージャーを戻す。

`ItsClass` この`VTypeData`のサブクラスを戻す。

`PutObjectSystem` このタイプ記述を作成したオブジェクトシステムをセットする。

オーバーライド可能な共用メンバー (Overidable Public Members)

`ConcreteTypec` このタイプの物理的インプレメンテーションを戻す。

`TypeCode` このタイプのタイプコードを戻す。

`Identical` 指定されたタイプがこのタイプと同じであるかどうかを決定する。

`SameLayoutAs` 指定されたタイプがこのタイプと同じレイアウトであるかどうかを決定する（名前は異なる場合があり得る）。

`CanAlwaysCast` 指定されたタイプが常にこのタイプにキャスト可能であるかどうかを決定する。

`CanSometimesCast` 指定されたタイプが時々このタイプにキャスト可能であるかどうかを決定する。

`Alignment` このタイプによって要請されるアラインメントを戻す。

`SizeOf` このタイプのインスタンスのサイズを戻す。

`Construct` 指定されたタイプの正しくキャストされたコピーであるこのタイプの新規なインスタンスを作成する。

`Cast` 指定されたタイプのインスタンスをこのタイプにキャストすること。

`Empty` このタイプのインスタンスを空にすること。

`Discard` このタイプのインスタンスを廃棄すること。

保護されたデータメンバー

`VObjectSystem itsObjectSystem` このタイ

(170)

プ

記述を作成したオブジェクトシステム。

タイプの定義

VTypeRefList

VTypeData オブジェクトの VrRefList

VTypeRef

VTypeData オブジェクトへの VrReference

Put BaseType

void PutBaseType (VTypeData \*base)

base これがベースとするタイプ。

Put BaseTypeは、この導出タイプがベースとするタイプをセットする。例えば、このタイプが符号無しショートに対するポインタである場合には、BaseTypeは、KVTypeUnsignedShortに関するタイプオブジェクトを戻さねばならない。このタイプが他のタイプから直接導出されていない場合には、このタイプはNULLを戻さねばならない。たとえば、これは、構造に定義に関する場合である。VcrAnyのインスタンスは、記憶装置内に配置された場合に実際に存在するタイプを戻さねばならない（すなわち、任意のCORBA及びOEL 2.0 VARIANTの両方は、特定のフィールドレイアウトを持つ構造のインスタンスである）。この方法は、ベースタイプの生成に関して未知である場合、当該ベースタイプをコールするためにサブクラスによってコールされることが可能である。

PutTypeManager

void PutTypeManager (VTypeManager \*manager)

manager このタイプを所有するタイプマネージャ

PutTypeManagerは、このタイプ記述を管理するVTypeManagerインスタンスをセットする。これは、タイプマネージャがその生成後において未知である場合に、当該タイプマネージャをベースクラスに記憶するためにサブクラスによって使用される。

GetActualInstance

```
void GetActualInstance (  const VcrDataRef  &instance,
                          VcrDataRef      &final)
```

(171)

`instance` 変換するためのインスタンス。

`final` (戻し) 変換後のインスタンス。

`GetActualInstance`は、できるだけ多く処理することの出来るタイプのインスタンスをとる。すなわち、これは、最後の結果を発見するために`VcrAlias`及び`VcrAny`タイプを処理することを意味する。インスタンスが`VcrAny`である場合には、`TypeOf`及び`VcrAny`の`Value`メンバーは`final`に格納される。`VcrAlias`のインスタンスは、ちょうど、別名の`BaseType`を用いて処理されたタイプを持つ。この方法は、全てのエイリアス及びエニーが処理されるまで、これらのオペレーションを実施する。この方法は、キャストの実施を支援するため

に用いられる。

`BaseType`

`VTypeData *BaseType()`

`BaseType`は、この導出タイプがベースとするタイプを戻す。例えば、このタイプが符号無しショートに対するポインタである場合には、`BaseType`は`KVTypeUnsignedShort`に関するタイプオブジェクトを戻さねばならない。このタイプが他のタイプから直接導出されない場合には、`NULL`を戻さねばならない。たとえば、これは、構造定義の場合である。

`TypeManager`

`VTypeManager *TypeManager()`

`TypeManager`は、このタイプ記述を管理する`VTypeManager`インスタンスを戻す。

`ItsClass`

`VClass *ItsClass()`

`ItsClass`は、このインスタンスのタイプを戻す。これは、`VTypeData`サブクラスのランタイム識別を実施するために用いられる。`VClass`の方法は、これが`TypeData`のどのサブクラスであるかを識別するために使用することが出来る。`ItsClass`は、変換中のタイプに関する情報を

(172)

収集するためにタイプ変換ルーチンにおいてコールされる。

**PutObjectSystem**

**void PutObjectSystem (VtObjectSystem system)**

**system** 当該タイプを所有するオブジェクトシステム。

**PutObjectSystem**は、このタイプを所有するオブジェクトシステムをセットする。この方法は、当該タイプ作成の直後においてのみ、オブジェクトシステムによってコールされなければならない。

**ConcreteType**

**VTypeData \*ConcreteType ()**

**ConcreteType**は、このタイプの物理的レイアウトであるタイプを戻す。例えば、CORBAシーケンスであるタイプは、3つのエレメントを含む構造である具象タイプを持つ。それら自体が具象であるような全てのタイプは、この方法からNULLを戻さなければならない。デフォルトインプレメンテーションはNULLを戻す。

**TypeCode**

**VTTypeCode TypeCode ()**

**TypeCode**は、このタイプに関するタイプコードを戻す。タイプコードは、2つの異なるタイプの同等性を決定するために、これらと比較するために使用できる。タイプマネージャは、タイプの両立性を決定するために使用される。TypeCodeへの初期コールに関して、タイプマネージャは、このタイプに対するVTTypeCodeを発見するためにコールされる。初期コールの後で、結果は当該タイプに記憶される。

**Identical**

**bool\_t Identical (VTypeData \*type)**

**type** 比較するためのタイプ。

**Identical**は、指定されたタイプが現在のタイプと同じであるかどうかを決定する。当該タイプが正確な複製であるならば、これは、真を戻さねばならない。全てのサブタイプ、サブフィールド、及び、名前はマッチしなければならない。

らない。ヘルプ情報は比較してはならない。タイプマネージャーは、2つのタイプが同じタイプコードを与えられることが可能であるかどうかを決定するために、この方法をコールする。ベースクラスインプレメンテーションは、2つのタイプが `TypeData` の同じサブクラスであることをチェックする。システムは2つのタイプの比較に際して、タイプ定義における循環性が当該システム停止の原因とならないことを、導出タイプが確認しなければならない。

#### `SameLayoutAs`

```
bool_t SameLayoutAs (VTypeData *type) = 0
```

`type` 比較するためのタイプ。

`SameLayoutAs` は、指定されたタイプが現在のタイプにレイアウトに関して同じであるかどうかを決定する。全てのサブタイプ、サブフィールド、及び、オフセットがマッチするならば、これは真を戻さねばならない。名前およびヘルプ情報は比較してはならない。この方法は、キャスト可能性を決定するためにコールされる。

#### `CanAlwaysCast`

```
bool_t CanAlwaysCast (VTypeData *type) = 0
```

`type` 比較するためのタイプ。

`CanAlwaysCast` は、全ての状況において、情報の損失なしに、タイプがこのタイプにキャストされることが可能であるかどうかを決定する。たとえば、ショートは常にロングに対してキャストされることが可能であるが、ロングの或るインスタンスに限り、ショートに対してキャストされることが可能である。これは、ショートからロングの

場合に限り、真を戻すはずである。

#### `CanSometimesCast`

```
bool_t CanSometimesCast (VTypeData *type) = 0
```

`type` 比較するためのタイプ。

或る種の状況または全ての状況において、情報の損失を生じることなしに、タイプがこのタイプに対してキャストされることが可能であるかどうかを、`Can`

(174)

`SometimesCast`が決定する。たとえば、ショートは常にロングに対してキャストされることが可能であるが、ロングの或るインスタンスに限り、ショートに対してキャストされることが可能である。これは、ショートからロングへ、又は、ロングからショートへ、いずれの場合にも、真を戻すはずである。

### `Alignment`

`long Alignment () = 0`

アラインメントは、このタイプのインスタンスのために必要とされるアラインメントを戻す。アラインメントはバイト数において指定する。例えば、8バイトダブルは、4バイト境界上にアラインされることが可能であるはずであり、この場合には、この方法は4を戻すはずである。可能な最低値は1である。

### `SizeOf`

`long SizeOf(void *instance) = 0`

`instance` サイズを決定するために使用されるインスタンスに対するポインタ。

`SizeOf`は、バイト表現されたタイプの特定インスタンスのサイズを戻す。インスタンスポインタがNULLである場合に、サイズが固定されているならば、`SizeOf`は、タイプの全てのインスタンスのサイズを戻さねばならない。インスタンスのサイズが可変である場合には、`SizeOf`は0または負を戻さねばならない。

### `Construct`

`long Construct ( const VcrDataRef &original,`

`original` コピーのもととなるオリジナルオブジェクトに対するタイプ及びポインタ。

`instance` (戻し) このタイプの新規なインスタンス。

`Construct`は、入力オブジェクトをこのタイプにキャストすることにより、新規なインスタンスを作成する。`Construct`は、戻された新規オブジェクトに対してバッファを割当てなければならない。戻し値は、新規なインスタンスのサイズでなくて

(175)

はならない。オリジナルインスタンスがこのタイプに対してキャストされないか、または、メモリの割当が不可能である場合には、`Construct`は0を戻さねばならない。

`Cast`

```
long Cast (  const VcrDataRef  &original,
            void              *instance) = 0
```

`original` コピーのもととなオリジナルオブジェクトに対するタイプ及びポインタ。

`instance` インスタンスを記憶する場所に関するポインタ。

`Cast`は、このタイプに入力オブジェクトをキャストすることにより、このタイプの新規なインスタンスを作成する。`Cast`は、新規なタイプインスタンスにバッファを割当ててはならないが、その代わりに、当該タイプに適合するだけ十分に大きいことが保証されている供給されたバッファを使用しなければならない。戻し値は、新規なインスタンスのサイズでなくてはならない。オリジナルインスタンスがこのタイプに対してキャストされることが出来ないか、又は、メモリーが割当てられることが出来ない場合には、`Cast`は0を戻さねばならない。このルーチンにパスされたインスタンスバッファは不要情報を含むことがあり得るので、完全に重ね書きされなければならない。

`Empty`

```
void Empty (void *instance) = 0
```

`instance` 空にされるためのタイプのインスタンス。

`Empty`は、当該タイプのインスタンスの内容を空にする。これは、当該インスタンス内に割当てられた全てのメモリーを解放しなければならないが、インスタンス自体を解放してはならない。

`Discard`

```
void Discard (void *instance) = 0
```

`instance` 空にされるためのタイプのインスタンス。

`Discard`は、このタイプのインスタンスの内容を空にする。これは、当該インスタンス内に割当てられた全てのメモリーを解放しなければならないが、イ

(176)

ンスタンス自体を解放してはならない。

### クラスVcrFundamental

VTypeDataの共用サブクラス

VcrFundamentalは基本タイプを定義する。このクラスは、全ての基本タイプのインプレメンテーションを提供するために、内部的にサブクラス化される。このク

ラスが他の基本タイプを提供しない限り、ユーザーは、通常、このクラスをサブクラス化する必要はない。

共用メンバー

VcrFundamental VcrFundamentalオブジェクト  
を作成する。

VcrFundamentalコンストラクタ

```
VcrFundamental( VTypeManager *manager,
                 VAtomRef      name,
                 VTTypeCode    type)
```

manager このオブジェクトによって用いられるタイプマネージャ。

name このポインタタイプの名前。

type この基本タイプのタイプコード。

例えば符号付きインスタンス等のような基本タイプを表すVcrFundamentalオブジェクトを作成する。

### クラスVcrAlias

VTypeDataの共用サブクラス

VcrAliasは他のタイプのエイリアス（別名）を定義する。このクラスのインスタンスは、一般に、クラスレジストリ用として使用する目的で、基本タイプの特殊バージョンに付加的ミクシン（mixin）を加えるためにのみ作成される。たとえば、符号無しロングは、特殊化されたカラーエディタを展開するためのmixinサポートを持つと言う点以外は符号無しロングに同じである「カラー」と命名されたエイリアスを持つことができる。

(177)

共用メンバー

`VcrAlias` `VcrAlias`オブジェクトを作成する。

`VcrAlias`コンストラクタ

```
VcrAlias( VAtomRef    name,
          VTypeData   *realType)
```

`name` このエイリアスの名前。

`realType` これが表すタイプ。

他のタイプのエイリアスを表す`VcrAlias`オブジェクトを作成する。全ての方法のインプレメンテーションは、`realType`オブジェクトの方法を適宜コールする。`mixIn`サポート用の方法は、`realType`上に直接マップされない方法に限られる。

クラス`VcrPointer`

`VTypeData`の共用サブクラス

`VcrPointer`は、他のタイプに対するポインタを定義する。

共用メンバー

`VcrPointer` `VcrPointer`オブジェクトを作成する。

`VcrPointer`コンストラクタ

```
VcrPointer( VAtomRef    name,
            VTypeData   *baseType)
```

`name` このポインタタイプの名前。

`baseType` これがポインタの対象とするタイプ。

他のタイプに対するポインタを表す`VcrPointer`オブジェクトを作成する。これは、オブジェクトに対するポインタ用に使用されてはならず、その代りに、`VcrObjectRef`を使用しなければならない。ファンクションに対するポインタは、`VcrFunctionPtr`によって実行されなければならない。

クラス`VcrObjectRef`

`VTypeData`の共用サブクラス

(178)

`VcrObjectRef`は、特定のオブジェクトタイプのインスタンスに関するリファレンス（ポインタ）を定義する。

共用メンバー

`VcrObjectRef` `VcrObjectRef`オブジェクトを作成する。

`Class` このタイプのインスタンスが参照するオブジェクトのクラスを戻す。

`VcrObjectRef`コンストラクタ

```
VcrObjectRef(VType Manager *manager,
              VAtomRef      name,
              VClassData    *refClass)
```

`manager` このオブジェクトによって用いられるタイプマネージャ。

`name` このリファレンスタイプの名前。

`refClass` 参照されたオブジェクトのクラス。

指定されたクラスのインスタンスに関するリファレンス（ポインタ）を表す `VcrObjectRef` オブジェクトを作成する。

`Class`

```
VClassData *Class()
```

`Class`は、このタイプのインスタンスが参照するオブジェクトのクラスを戻す。

クラス `VcrFunctionPtr`

`VTypeData`の共用サブクラス

`VcrFunctionPtr`は、指定されたシグネチャを用いて、ファンクションに対するポインタを定義する。

共用メンバー

`VcrFunctionPtr` `VcrFunctionPtr`オブジェクトを作成する。

`Function` ファンクションシグネチャ記述を戻す。

(179)

**VcrFunctionPtr**コンストラクタ

```

VcrFunctionPtr( VTypeManager    *managerf,
                 VAtomRef        name,
                 VFunctionData    *funcSig)

```

**manager** このオブジェクトによって用いられるタイプマネージャ。

**name** このファンクションポインタタイプの名前。

**funcSig** これがポイントするファンクションのシグネチャ記述。

指定されたシグネチャを用いて、ファンクションに対するポインタを表す**VcrFunctionPtr**オブジェクトを作成する。**VFunctionData**の所有権は、このオブジェクトに譲渡される。この**VFunctionData**は、他の場所において使用されてはならない。

**Function**

```
VFunctionData *Function()
```

**Function**は、このタイプのインスタンスがポイントするファンクションの記述を返す。

**クラスVcrStructItem**

**VcrStructItem**は、構造または合併においてフィールドを定義する。これは、フィールドネーム、タイプ、及び、オフセットを構造内に含む。このクラスのインスタンスには、タイプコードを割り当てることが出来ない。

## 共用メンバー

**VcrStructItem VcrStructItem**オブジェクトを作成する。

**Name** このフィールドの名前を返す。

**Type** このフィールドのタイプを返す。

**Offset** このフィールドの構造内にオフセットを返す。

**VcrStructItem**コンストラクタ

(180)

```

VcrStructItem( VAtomRef  name,
                VTypeData *theType,
                long      offset)

```

`name` この構造フィールドの名前。

`theType` この構造フィールドのタイプ。

`offset` 構造内のこのフィールドのオフセット。

構造内のフィールドを表す `VcrStructItem` オブジェクトを作成する。

`Name`

```
VAtomRef Name ()
```

`Name` は、この構造または合併フィールドの名前を戻す。

`Type`

```
VTypeData *Type ()
```

`Type` は、この構造または合併フィールドのタイプを戻す。

`Offset`

```
long offset ()
```

`Offset` は、構造内のフィールドのオフセットを戻す。

クラス `VcrStruct`

`VTypeData` の共用サブクラス

`VcrStruct` は構造の定義を定義する。この構造は、当該構造の初めから固定した数のオフセットにおいて固定数のフィールドを含む。

共用メンバー

`VcrStruct VcrStruct` オブジェクトを作成する。

`AddItem` 構造にフィールドを加える。

`GetItemByName` 指定されたフィールドのフィールド記述を戻す。

`GetItemByIndex` 指定されたフィールドのフィールド記述を戻す。

`GetItemCount` 構造におけるフィールド数を戻す。

(181)

## VcrStruct コンストラクタ

```

VcrStruct( VtypeManager *manager,
           VAtomFef      name,
           VcrStructItem *item1=0,
           VcrStructItem *item2=0,
           ...
           VcrStructItem *item6=0)

```

**manager** このオブジェクトによって用いられるタイプマネージャ。

**name** この構造の名前。

**item1 - item6** 構造に加えるためのオプションのアイテム。

構造を表すVcrStructオブジェクトを作成する。フィールドは、そのレイアウトを定義するために、作成の後において、構造に加えることが出来る。

item1からitem6までのうちのいずれかが指定されている場合には、これらの各々に対してAddItemが順々にコールされたかのように、これらが構造定義に加えられる。

## AddItem

```
void AddItem(VcrStructItem *item)
```

**Item** 構造に加えるためのフィールド。

新規フィールドを構造に加えること。タイプ両立性およびキャストビリティを決定するにはフィールドを加える順序が用いられるので、フィールドを加える順序は重要である。

## GetItemByName

```
VcrStructItem *GetItemByName (VAtomRef name)
```

**name** 検索するためのフィールドの名前。

所定の名前のフィールドを構造から検索すること。

## GetItemByIndex

```
VcrStructItem *GetItemByIndex (long index)
```

**index** 検索するためのフィールドのインデックス。

構造における所定のインデックスのフィールドを構造から検索すること。インデックスは、ゼロから出発してフィールドが構造に加えられた順序に従って定義

(182)

される。

`GetItemCount`

`long GetItemCount()`

この構造におけるフィールド番号を戻す。

クラス `VcrEnumItem`

`VcrEnumItem`は、列挙タイプの可能な値のうちの1つである列挙定数を定義

する。このクラスのインスタンスには、タイプコードを割り当てることが出来ない。

共用メンバー

`VcrEnumItem` `VcrEnumItem`オブジェクトを作成する。

`Name` 列挙定数の名前を戻す。

`Value` 列挙定数の値を戻す。

`VcrEnumItem`コンストラクタ

`VcrEnumItem( VAtomFef name,  
VValue value)`

`name` この列挙定数の名前。

`value` この列挙定数の値

`enum` (列挙定数の集合) のうちの1つを表す `VcrEnumItem` オブジェクトを作成する。

`Name`

`VAtomRef Name()`

`Name` は、列挙定数の名前を戻す。

`Value`

`VValue Value()`

`Value` は、列挙定数の値を戻す。

クラス `VcrEnum`

`VTypeData` の共用サブクラス

(183)

`VcrEnum`は、列挙タイプを定義する。列挙タイプは多数の列挙定数を含む。この場合、各定数は関連した名前および値を持つ。

共用メンバー

`VcrEnum` `VcrEnum`オブジェクトを作成する。

`AddItem` `enum`に列挙定数を加える。

`GetItemByName` 指定された列挙定数の記述を返す。

`GetItemByIndex` 指定された列挙定数の記述を返す。

`GetItemCount` この`enum`における列挙定数の数を返す。

`VcrEnum`コンストラクタ

```

VcrEnum ( VAtomFef      name,
           VTypeData     *enumType,

           VcrEnumItem  *item1=0,
           VcrEnumItem  *item2=0,
           ...
           VcrEnumItem  *item6=0)

```

`manager` このオブジェクトによって用いられるタイプマネージャ。

`name` この列挙の名前。

`enumType` 列挙が格納されるタイプ。

`item1 - item6` 列挙に加えるためのオプションとしてのアイテム。

列挙を表す`VcrEnum`オブジェクトを作成する。定数の可能な値を定義するために、定数は、作成した後において、列挙に加えることが出来る。列挙のタイプは、整数基本タイプでなくてはならない。`item1`から`item6`までのうちのいずれかが指定されている場合には、これらの各々に対して`AddItem`が順々にコールされたかのように、これらが構造定義に加えられる。

`AddItem`

```

Void AddItem (VcrEnumItem *item)

```

`item` 列挙に加えるための定数。

新規な定数を列挙に加えること。タイプ両立性およびキャストビリティを決定するにはフィールドを加える順序が用いられるので、フィールドを加える順序は

重要である。

`GetItemByName`

`VcrEnumItem *GetItemByName (VAtomRef name)`

`name` 検索するための定数の名前。

その名前によって、列挙から列挙定数を検索すること。

`GetItemByIndex`

`VcrEnumItem *GetItemByIndex (long index)`

`index` 検索するための定数のインデックス。

列挙におけるそのインデックスにより、列挙から定数を検索すること。インデックスは、ゼロから開始して定数を列挙に加えた順序に従って定義される。

`GetItemCount`

`long GetItemCount ()`

この列挙における定数の数を戻す。

クラス `VcrSequence`

`VTypeData`の共用サブクラス

`VcrSequence`は抽象クラスである。これは、その長さをインスタンススペースによりインスタンスに関して決定しても差し支えない可変長アレイを定義する。例えば、ストリングは`VcrSequences`として実行される。同様に、`VcrStructSequence`は、`VcrSequence`のトップにおいて実行される。更に、固定長さのアレイも、`VcrArray`を用いて、`VcrSequence`のサブクラスとして記述される。

共用メンバー

`VcrSequence` `VcrSequence`オブジェクトを作成する。

`GetLowerBound` シーケンスの下限を戻す。

保護されたメンバー

`PutConcreteType` 作成した後において、具象タイプをセットする。

オーバーライド可能な共用メンバー

(185)

`GetLength` シーケンスのインスタンスの長さを戻す。

`GetMaxLength` シーケンスの最大長を戻す。

`GetElement` シーケンスの指定されたエレメントを戻す。

`IsLinear` シーケンスデータが線型であるかどうかを戻す。

`VcrSequence` コンストラクタ

```
VcrSequence (VAtomRef    name,
              VTypeData   *concreteType,
              VTyepData   *baseType,
              long         lbound)
```

`name` このシーケンスタイプの名前。

`concreteType` このシーケンスがそれによって作られたタイプ。

`baseType` これがそのシーケンスであるようなタイプ。

`lbound` シーケンスインデックスの下限。

他のタイプの他のインスタンスの可変長アレイに対するポインタを表す `VcrSequence` オブジェクトを作成する。

`GetLowerBound`

```
long GetLowerBound ()
```

許容されたシーケンスインデックスの下限を戻す。

`PutConcreteType`

```
void PutConcreteType (VTypeData *concreteType)
```

`concreteType` 実際のシーケンスタイプ。

`PutConcreteType` は、作成した後において、シーケンスの実際のタイプをセットする。このタイプは、構造等々の中に格納される実タイプである。たとえば、CORBAシーケンスは、3つのエレメントを含む構造である具象タイプを持つ。

`GetLength`

```
long GetLength (void *instance) = 0
```

`instance` このシーケンスタイプのインスタンス。

(186)

`GetLength`は、このシーケンスタイプの特定のインスタンスの長さを返す。これは、`VcrSequence`のサブクラスにおいて実行されなければならない。

`GetMaxLength`

```
long GetMaxLength (void *instance) = 0
```

`instance` このシーケンスタイプのインスタンス。

`GetMaxLength`は、このシーケンスタイプの特定のインスタンスに関する最大長を返す。これは、シーケンスアレイに関するデータ格納の割当てられた最大長を返す。これは、`VcrSequence`のサブクラスにおいて実行されなければならない。

`GetElement`

```
void *GetElement ( void *instance,
                   long index) = 0
```

`instance` このシーケンスタイプのインスタンス。

`index` 検索するためのエレメントのインデックス。

シーケンスの指定されたエレメントに対するポインタを返す。インデックスがシーケンスの正当な限度の外側にある場合には、これはNULLを返す。

`IsLinear`

```
bool_t IsLinear () = 0
```

シーケンスが、指定されたタイプの線型シーケンスであるかどうかを返す。戻しがTRUEである場合には、シーケンスの第2エレメントがメモリ内の第1エレメントに直接従う。`IsLinear`がFALSEを返す場合には、シーケンスはスパースであっても差し支えなく、そして、`GetElement`を介してのみアクセス可能である。

クラス `VcrArray`

`VcrSequence`の共用サブクラス

`VcrArray`は、固定した長さのアレイを記述する。固定した長さのアレイは、固定した上限および下限を持たねばならない。下限はゼロに制限されない

(187)

。

共用メンバー

VcrArray VcrArray オブジェクトを生成

VcrArrayコンストラクタ

```

VcrArray ( VAtomRef  name,
           VTypeData *elemType,
           long      lbound,

```

name このアレイトイプの名前。

elemType アレイにおけるエレメントのタイプ。

lbound アレイインデックスの下限。

len アレイの長さ。

アレイを表すVcrArrayオブジェクトを作成する。アレイは、ゼロであることを必要としない下限を持つ固定した長さである。

クラスVcrStructSequence

VcrSequenceの共用サブクラス

VcrStructSequenceは、length、長さ、最大長さ、及び、エレメントのアレイに対するポインタを含むメンバーを含む構造として定義されるシーケンスを表す。

共用メンバー

VcrStructSequence VcrStructSequence  
オブジェクトを作成する。

VcrStructSequenceコンストラクタ

```

VcrStructSequence ( VAtomRef  name,
                   VcrStruct  *concreteType,
                   VcrStructItem *length,
                   VcrStructItem *maxLen,
                   VcrStructItem *array,
                   long          lbound)

```

name このアレイトイプの名前。

concreteType シーケンス自体の物理的レイアウトのタイプ。

(188)

`length` シーケンスの長さを表す構造フィールド。

`maxLength` シーケンスの最大長を表すフィールド。

`array` シーケンスデータに対するポインタを表す構造フィールド。

`lbound` シーケンスインデックスの下限度。

長さ、最大長、及び、データに対するポインタのに関するエレメントを含む構造体として記述されるシーケンスを表す `VcrStructSequence` オブジェクトを作成する。このシーケンスのベースタイプは、`array->Type()` のベースタイプである。`array->Type()` は `VcrPointer` または `VcrReference` でなければならない。

### クラス `VcrString`

`VcrSequence` の共用サブクラス

`VcrString` はキャラクタの可変長ストリングを記述する抽象クラスである。各サブクラスは、そのデータをあらゆる適当なフォーマットにおいて記憶することが出来る。例えば、`CString` サブクラスは `NUL` で終了するキャラクタの線型アレイとしてストリングを格納する。`BasicString` は、キャラクタの線型アレイによって後続される或る長さにおいて格納される。

共用メンバー

`VcrString VCrString` オブジェクトを作成する。

`VcrString` コンストラクタ

<code>VcrString(</code>	<code>VAtomRef</code>	<code>name,</code>
	<code>VTypeData</code>	<code>*concreteType,</code>
	<code>VtypeData</code>	<code>*baseType,</code>
	<code>long</code>	<code>lbound</code>

`name` このストリングタイプの名前。

`concreteType` このストリングがそれから作成されるタイプ。

`baseType` スtringのタイプ。これは、符号付き又は符号なしキャラクタのいずれかでなくてはならない。

`lbound` スtringインデックスの下限。

ストリングを表すために用いられる `VcrString` オブジェクトを作成する。このクラスは、ストリングの異なるインプレメンテーションがこれに基づいて作成される抽象ベースクラスである。

### クラス `VcrAny`

`VTypeData` の共用サブクラス

`VcrAny` は、数種のタイプの 1 つの値を含むことができるデータタイプを定義する抽象クラスである。このタイプは、例えば `OLE2.0` の `VARIANT` 及び `CORBA`

のようなタイプを定義するために用いられる。`VcrAny` タイプは、変換されつつある値のタイプが両方のタイプに含まれていることを条件として、相互間において透過的に変換可能である。インスタンスが、事前に定義済みの固定した異なるタイプの集合に含まれる値を含むと言う制限を受けない点において、`VcrAny` はユニオンと異なる。

共用メンバー

`VcrAny` `VcrAny` オブジェクトを作成する。

保護されたメンバー

`PutConcreteType` 作成後において、具象タイプをセットする。

オーバーライド可能な共用メンバー

`Types` サポートされたタイプのリストを返す。

`TypeOf` 指定されたインスタンスのタイプを返す。

`Value` 指定されたのインスタンスからの実値を返す。

`VcrAny` コンストラクタ

```
VcrAny( VAtomRef  name,
        VTypeData  *concreteType)
```

`name` このタイプの名前。

`concreteType` 任意のタイプの具象タイプ。

ファンクションからのパスおよび戻りをサポートするために用いられる。他の

(190)

タイプの任意の集合を含むことの出来るタイプを表す `V c r A n y` オブジェクトを作成する。これは、例えば任意のタイプおよび `V A R I A N T` のような総称タイプを定義するためにサブクラス化されなければならない抽象ベースクラスである。

`P u t C o n c r e t e T y p e`

`void PutConcreteType (VTypeData *concreteType)`

`c o n c r e t e T y p e` 任意の実タイプ。

`P u t C o n c r e t e T y p e` は、作成後において、任意の実タイプをセットする。このタイプは構造体等々の中に格納可能な実タイプである。たとえば、任意の `C O R B A` は、2つのエレメントを含む構造体である具象タイプを持つ。

`T y p e s`

`V T T y p e L i s t T y p e s () = 0`

`T y p e s` は、この総称タイプに含まれることが可能なタイプのリストの部分集合を返す。当該タイプが他の任意のタイプを含むことができる場合には、これは、空のリストを

戻さねばならない。サポートされるタイプの集合が完全に動的である場合には、空のリストが戻されなければならない。このリストは、インスタンスがどのようなタイプを含んでいる可能性があるかに関するヒントとみなさなければならない。

`T y p e O f`

`V T y p e D a t a * T y p e O f ( v o i d * i n s t a n c e ) = 0`

`I n s t a n c e` 質問するためのインスタンス。

`T y p e O f` は、この総称タイプのインスタンスに含まれる値のタイプを返す。当該タイプがタイプ管理システムに関して記述されることが出来ない場合には、`T y p e O f` は `N U L L` を戻さねばならない。

`V a l u e`

`void *Value (void *instance) = 0`

`i n s t a n c e` 質問のためのインスタンス

(191)

`Value`は、この総称タイプに含まれる値に対するポインタを検索する。インスタンスが値を含まない場合には、これは、`NUL L`を戻しても差し支えない。

### クラス `VcrUnionItem`

`VcrStructItem`の共用サブクラス

`VcrUnionItem`はユニオンにおけるフィールドを定義する。これは、フィールドネーム、タイプ、スイッチ値、及び、オフセットをユニオン内に含む。オフセットは、大抵の機械アーキテクチャに関して、一般に、ゼロである。このクラスのインスタンスにはタイプコードは割り当てられない。

共用メンバー

`VcrUnionItem VcrUnionItem`オブジェクトを作成する。

`Value` これが使用中の現行フィールドであることを指定する値を戻す。

`VcrUnionItem`コンストラクタ

```
VcrUnionItem(  VAtomRef  name,
                VTypeData *theType,
                VValue    value,
                long      offset = 0)
```

`name` このユニオンフィールドの名前。

`theType` このユニオンフィールドのタイプ。

`value` このフィールドのスイッチ値。

`offset` ユニオン内におけるこのフィールドのオフセット（0でない場合）。

ユニオン内におけるフィールドを表す `VcrUnionItem` オブジェクトを作成する。このフィールドが「デフォルト」フィールドであるべきである場合に、他のフィールドに関して一切の値マッチが発見されないならば、値はタイプ `KVTypeVoid` でなくてはならない。

`Value`

**VValue Value ()**

**V a l u e**は、このフィールドに関するスイッチ値の値を戻す。ユニオンとペアを構成するスイッチアイテムがこの値である場合には、当該ユニオンは、このフィールドによって表される値を含む。

**クラスVcrUnion**

**V c r A n y**の共用サブクラス

**V c r U n i o n**は抽象ユニオンタイプ定義を定義する。これは、ユニオンの全ての異なるレイアウトに関して共通ベースを可能にすることに役立つ。たとえば、**V c r S t r u c t U n i o n**は、それ自体を構造対として定義するユニオンである。この構造は、埋め込まれた構造体およびスイッチフィールドを持つ。それ自身がスイッチフィールドを持たない他のタイプのユニオンであっても差し支えない。実スイッチフィールドは、**V c r U n i o n**のこのサブクラスに関して、含まれるタイプにおける当該ユニオンの外側に所在しても差し支えない。

共用メンバー

**V c r U n i o n** **V c r U n i o n**オブジェクトを作成する。

**V c r U n i o n**コンストラクタ

```
VcrUnion( VAtomRef    name,
          VTypeData  *concreteType)
```

**name** このユニオンの名前。

**concreteType** このユニオンの具象タイプ。

ユニオンを表す**V c r U n i o n**オブジェクトを作成する。具象タイプは、ユニオン自体の実レイアウトを定義する構造である。

**クラスVcrStructUnion**

**V c r U n i o n**の共用サブクラス

**V c r S t r u c t U n i o n**はCORBAスタイルユニオンタイプ定義を定義する。ユニオンは、スイッチフィールドを含む構造、ユニオンのどのサブフィールドを使用するかを決定するフィールド、及び、実ユニオンのフィールドを含むネストされた構造を定義

(193)

することによって作成される。これらのフィールドは、`VcrStructItem`の代わりに`VcrUnionItem`のインスタンスでなければならない。

共用メンバー

`VcrStructUnion` `VcrStructUnion`オブジェクトを作成する。

`VcrStructUnion`コンストラクタ

```
VcrStructUnion( VAtomRef    name,
                 VTypeData    *concreteType,
                 VcrStructItem *switchItem,
                 VcrStructItem *unionItem)
```

`name` このユニオンの名前。

`concreteType` このユニオンの具象タイプ。

`switchItem` 現行ユニオンが、どのフィールドを含むかを決定する構造におけるアイテム。

`unionItem` ユニオンエレメントを含むネストされた構造体としての構造におけるアイテム。

ユニオンを表す`VcrStructUnion`オブジェクトを作成する。具象タイプは、ユニオン自体の実レイアウトを定義する構造である。

クラス`VcrOpaque`

`VTypeData`の共用サブクラス

`VcrOpaque`は、未知の定義のタイプを定義する抽象クラスである。このクラスは、既知のタイプにインスタンスをキャストし、更に、この不透明なタイプに既知のタイプをキャストすることを可能にする方法を提供する。`VcrOpaque`サブクラスは、可能な基本タイプおよび導出タイプの集合によって記述することの出来ないタイプマネージャーにタイプを導入するために用いられる。`VcrOpaque`のサブクラスは、その内部表現に対して他のタイプをキャスト可能でなくてはならない。

共用メンバー

`VcrOpaque` `VcrOpaque`オブジェクトを作成する。

保護されたメンバー

`PutConcreteType` 作成後において、具象タイプをセットする。  
。

オーバーライド可能な共用メンバー

`CastToList` これがキャストすることのできるタイプのリストを返す。

`CastFromList` このタイプにキャストされることのできるタイプのリストを返す。

`CastAlwaysReverseCast` このタイプが指定されたタイプに常にキャストされることができかどうかを決定する。

`CansometimesReverseCast` このタイプが指定されたタイプに時々キャストされることができかどうかを決定する。

`ReverseConstruct` このタイプのインスタンスの正しくキャストされたコピーである指定されたタイプのインスタンスを作成する。

`ReverseCast` このタイプのインスタンスを指定されたタイプにキャストする。

`VcrOpaque` コンストラクタ

```
VcrOpaque ( VAtomRef    name,
             VTypeData   *concreteType)
```

`name` このタイプの名前。

`concreteType` この不透明タイプの具象タイプ。ファンクションに対するパス及び戻しをサポートするために用いられる。

`Vcropaque` オブジェクトを作成する。このタイプのサブクラスは、全ての仮想方法に対してインプレメンテーションを提供する。この抽象クラスは、単に不透明タイプをタイプ管理システムに統合するためのフレームワークを定義

する。

`PutConcreteType`

`void PutConcreteType (VTypeData *concreteType)`

`concreteType` 実際に不透明なタイプ。

生成後において、`PutConcreteType`は、不透明な実タイプをセットする。このタイプは、構造内に格納されるか、ファンクションにパスされる実タイプである。このタイプは、これらの不透明オブジェクトをアーギュメントとしてパスし、そして、これらを戻しとして受け取るためにスタック上に正しく配置することを管理するために必要とされる。

`CastToList`

`VTypeRefList CastToList () = 0`

`CastToList`は、このタイプのあらゆるインスタンスが情報の損失なしで投げられ得るタイプの部分的なリストを戻す。このタイプのリストは、このタイプのインスタンスをどのように処理するかを決定するために言語インタプリタに与えられる心得として

使用できる。

`CastFromList`

`VTypeRefList CastFromList () = 0`

`CastFromList`は、情報の損失なしのこのタイプのインスタンスにキャストされ得るこれらのタイプの部分的なリストを戻す。これらのタイプのこのリストは、このタイプのインスタンスをどのように処理するかを決定するために言語インタプリタに与えられる心得として使用できる。

`CanAlwaysReverseCast`

`bool_t CanAlwaysReverseCast (VTypeData *type) = 0`

`type` 比較するためのタイプ。

`CanAlwaysReverseCast`は、全ての状況において、情報の損失なしに、指定されたタイプが現行タイプにキャストすることが可能であるかどうかを決定する。これは、`CanAlwaysCast`の逆方向である。たと

(196)

例えば、ショートは常にロングにキャストできるが、ショートにキャストすることのできるは或る種のロングのインスタンスに限られる。これは、ショートからロングの場合に限り真を戻す。

`CanSometimesReverseCast`

`bool_t CanSometimesReverseCast (VTypeData *type) = 0`

`type` 比較するためのタイプ。

`CanSometimesReverseCast`は、或る種のまたは全ての状況において、情報の損失なしに、指定されたタイプが現行タイプにキャスト可能であるかどうかを決定する。これは、`CanSometimesCast`の逆方向である。たとえば、ショートは常にロングにキャストできるが、ショートにキャストすることのできるは或る種のロングのインスタンスに限られる。これは、ショートからロング、又は、ロングからショートへキャストするいずれかの場合に真を戻す。

`ReverseConstruct`

`long ReverseConstruct ( void *instance,  
VcrDataRef &to) = 0`

`instance` このタイプのインスタンス。

`to` (入力および戻し) 作成されるべき最後オブジェクト、又は、戻しに際して新規に作成されるアイテムに対するポインタのタイプ。

`ReverseConstruct`は、このタイプの入力オブジェクトを要求されたタイプにキャストすることにより指定されたタイプの新規なインスタンスを作成する。`Co`

`nstruct`と比較した場合、これは、作成の方向の反対方向を実施する。`ReverseConstruct`は、新規に戻されたオブジェクトに対して、バッファを割当てなければならない。戻し値は、新規インスタンスのサイズでなくてはならない。オリジナルのインスタンスがこのタイプにキャストされることが出来ないか、或いは、メモリを割当てられることが出来ない場合には、`ReverseConstruct`は0を戻さねばならない。

(197)

## ReverseCast

```
long ReverseCast ( void      *instance,
                   VcrDataRef &to) = 0
```

`instance` このタイプのインスタンス。

`to` 充填するためのインスタンスのタイプ及び充填するためのインスタンスに対するポインタ。

`ReverseCast`は、このタイプの入力オブジェクトを指定されたタイプにキャストすることによって指定されたタイプの新規なインスタンスを作成する。`ReverseCast`は、`Cast`との比較に際して、反対方向のキャストを実施する。`ReverseCast`は、新規なタイプインスタンスに対して、バッファを割当ててはならないが、その代りに、当該タイプに充分適合できる程度に大きいことが保証される供給されたバッファを使用しなければならない。戻し値は、新規なインスタンスのサイズでなくてはならない。オリジナルのインスタンスがこのタイプにキャストされることが出来ないか、或いは、メモリを割当てられることが出来ない場合には、`ReverseCast`は0を戻さねばならない。このルーチンにパスされるインスタンスバッファは不要情報を含むこともあり得るので、完全に重ね書きされなければならない。

## クラスVTypeManager

VPrimaryの共用サブクラス

`VTypeManager`はタイプ記述を管理する。このクラスは、タイプ記述を記憶し、そして、一意的な各タイプ定義に関して一意的な`id` (`VTypeCodes`)を提供する。一般に、1つのプロセスにつき1つのタイプマネージャが存在する。

共用メンバー

`VTypeManager` `VTypeManager`オブジェクトを作成する。

`Register` 新規なタイプを登録し、そして、その`VTypecode`を戻す。

`TypeDestroyed` タイプオブジェクトが削除済みであることをタ

(198)

イプマネージャーに通告する。

**RegisterPersistent** 持続的なタイプコードによってタイプを登録する。

**Lookup** そのVTypecodeによりタイプの記述を戻す。

**SameLayout** 2つのタイプが同等であるかどうかを決定する（サブエレメントの名前は異なることもあり得る）。

**CanAlwaysCast** 第1のタイプは第2のタイプへ常にキャストできるかどうかを決定する。

**CanSometimesCast** 第1のタイプは第2のタイプに時々キャストできるかどうかを決定する。

**Sizeof** 指定されたタイプのインスタンスのサイズを決定する。

**Construct** 他のタイプとして1つのタイプのコピーを作成する。

**Cast** 1つのタイプのインスタンスを他のタイプにキャストする。

**Empty** 指定されたタイプのインスタンスを空にする。

**Discard** 指定されたタイプのインスタンスを廃棄する。

**TypeName** 指定されたタイプの名前を戻す。

**TypeHelp** 指定されたタイプのヘルプ情報を戻す。

**VTypeManager** コンストラクタ

**VTypeManager()**

**VTypeManager** オブジェクトを作成する。作成後において、タイプマネージャーは、キャストを実行し、そして、タイプ記述を受け取って、記憶する準備が整った状態にある。

**Register**

**VTypeCode Register (VTypeData \*type)**

**type** 登録するためのタイプ

**Register** は、タイプマネージャーにより、新規なタイプを登録する。このタイプが既存のタイプと同じであることが (**VTypeData::Identical** を用いて) 決定された場合には、このタイプ記述には、同等である

既存のタイプと同じ `VTTypeCode` が与えられる。これは、タイプコードに関する第1のリクエストに応じて、これらのタイプのインスタンスによって自動的にコールされる。それを直接コールする理由はない。

## TypeDestroyed

```
void TypeDestroyed (VTypeData *type)
```

t y p e 破壊されつつあるタイプ。

`TypeDestroyed`は、タイプが破壊されたことをタイプマネージャに通告す

る。これは、タイプ同等性を決定する際に使用された記憶済みの情報を、タイプマネージャが浄化して、除去することを可能にする。マネージャが当該タイプのフィールドを引用する必要がある場合があり得るので、タイプオブジェクトが実際に自由になる以前に、これがコールされていなければならない。タイプに関する全てのリファレンスが消失した後ではあるが、当該タイプを削除する削除する以前に、`VTypeData`はこれを自動的にコールする。

## Register Persistent

```
RegisterPersistent (VTypeCode    tcode,
                   VTypeData    *type)
```

`tcode` このタイプに関する持続的なタイプコード。

type 登録するためのタイプ記述

RegisterPersistentは、タイプマネージャにより、タイプ記述を登録する。タイプ記述には、指定されたタイプコードが与えられる。これらのタイプコードは、Visual Edgeによって割当てられなければならない。この方法はRegisterの代りに用いられるので、事前定義済みのタイプをタイプマネージャに加えることができる。持続性のタイプオブジェクトは、一旦、タイプマネージャーに加えられると、当該タイプマネージャが存在する限り、存続する。基本タイプオブジェクトは、作成に際してこの方法を自動的にコールするので、この方法を直接コールする必要はない。

## Lookup

(200)

VTypeData \*Lookup(VTTypeCode type)

type    ルックアップするためのタイプ。

Lookupは、指定されたタイプコードに関するタイプ定義を戻す。このタイプが基本タイプであるか、或いは、このタイプマネージャにおいて現在未だ定義されていない場合には、NULLが戻される。

SameLayout

bool\_t SameLayout ( VTTypeCode type1,  
VTTypeCode type2)

type1    比較用に使用するための第1のタイプ。

type2    比較用に使用するための第2のタイプ。

SameLayoutは、指定されたタイプのレイアウトが同じであるかどうかを決定する。全てのサブタイプ、サブフィールド、及び、オフセットがマッチする場合には、これは、真を戻さなければならない。名前とヘルプ情報は比較してはならない。この方法は、キャストする可能性を決定するためにコールされる。

CanAlwaysCast

bool\_t CanAlwaysCast ( VTTypeCode type1,  
VTTypeCode type2)

type1    比較用に用いるための第1のタイプ。

type2    比較用に用いるための第2のタイプ。

CanAlwaysCastは、全ての状況において、情報の損失なしに、第1のタイプを第2のタイプにキャストすることができるかどうかを決定する。たとえば、ショートは常にロングにキャスト可能であるが、ショートにキャストできるのは或る種のロングのインスタンスのみに限られる。これは、ショートからロングの場合にのみ真を戻す。

CanSometimesCast

bool\_t CanSometimesCast ( VTTypeCode type1,  
VTTypeCode type2)

(201)

`type1` 比較用に用いるための第1のタイプ。

`type2` 比較用に用いるための第2のタイプ。

`CanSometimesCast`は、或る種の状況または全ての状況において、情報の損失なしに、第1のタイプが第2のタイプにキャスト可能であるかどうかを決定する。たとえば、ショートは常にロングにキャストできるが、ショートにキャストすることのできるは或る種のロングのインスタンスに限られる。これは、ショートからロング、又は、ロングからショートへキャストするいずれかの場合に真を返す。

`Alignment`

`long Alignment (VTttypeCode type)`

`type` アラインするためのタイプ。

`Alignment`は、このタイプのインスタンスのために必要とされるアラインメントを返す。アラインメントはバイト数によって指定される。例えば、8バイトダブルは、4バイトの境界上に配列することが可能であり、この場合、この方法は4を返す。可能な最低値は1である。

`SizeOf`

`long SizeOf( VTTypeCode type,  
void *instance = 0)`

`type` 当該インスタンスのタイプ

`instance` サイズ決定用のインスタンスに対するポインタ。

`SizeOf`は、当該タイプの特定のインスタンスのサイズを返す。インスタンスポイ

ンタがNULLである場合において、サイズが固定されている場合には、`sizeOf`は、当該タイプの全てのインスタンスのサイズを返さねばならない。インスタンスのサイズが可変である場合には、`SizeOf`は0或いは負を返さねばならない。

`Construct`

(202)

```

long Construct ( VTTypeCode  origtype,
                 void         *original,
                 VTTypeCode  newtype,
                 void         **instance)

```

`origtype` コピー用に用いるためのオリジナルオブジェクトのタイプ。  
。

`original` コピーしようとするオリジナルオブジェクトに対するポインタ。

`newtype` ターゲットオブジェクトのタイプ。

`instance` (戻し) ターゲットタイプのインスタンス。

`Construct`は、入力オブジェクトをターゲットタイプにキャストすることにより、ターゲットタイプの新規なインスタンスを作成する。`Construct`は、新規なオブジェクトに対してバッファを割当てなければならない。戻しの値は、新規なインスタンスのサイズでなくてはならない。オリジナルインスタンスをターゲットタイプにキャストすることが出来ないか、或いは、メモリーがの割当てが不可能である場合には、`Construct`は0を戻さねばならない。

`Cast`

```

long Cast ( VTTypeCode  origtype,
            void         *original,
            VTTypeCode  totype,
            void         *instance)

```

`origtype` コピー用に用いるためのオリジナルオブジェクトのタイプ。  
。

`original` コピーしようとするオリジナルオブジェクトに対するポインタ。

`totype` ターゲットのタイプ。

`instance` 当該インスタンスを格納する場所に対するポインタ。

`Cast`は、入力オブジェクトをターゲットタイプにキャストすることにより

(203)

、ターゲットタイプの新規なインスタンスを作成する。C a s tは、新規なタイプインスタンスにバッファを割当ててはならないが、その代りに、当該タイプに充分適合するだけ大きいことが保証されている供給されたバッファを使用しなければならない。戻し値は、新規なインスタンスのサイズでなくてはならない。オリジナルインスタンスをターゲットタイプにキャストすることが出来ないか、又は、メモリーの割当てが不可能である場合には、C a s tは0を戻さねばならない。このルーチンにパスされたインスタンスバッファは不要情報を含む可能性があるため、完全に重ね書きしなければならない。

### Empty

```
void Empty ( VTTypeCode  type,
             void          *instance)
```

t y p e インスタンスのタイプ。

i n s t a n c e 空にされるタイプのインスタンス。

Emptyは、当該タイプのインスタンスの内容を空にする。これは、当該インスタンス内に割当てられたメモリー全てを解放しなければならないが、インスタンス自身を解放してはならない。インスタンス自身を解放するためには、代わりに、D i s c a r dをコールしなければならない。

### Discard

```
void Discard (VTTypeCode  type,
              void          *instance)
```

t y p e インスタンスのタイプ。

i n s t a n c e 廃棄されるべきタイプのインスタンス。

D i s c a r dは、タイプのインスタンスの内容を空にする。これは、インスタンス内に割当てられた全てのメモリーを解放し、そして、インスタンス自体を解放する。

### TypeName

```
VAtomRef TypeName (VTTypeCode type)
```

t y p e それについての情報を入手しようとするタイプ。

`Type Name`は、指定されたタイプの名前を検索する。当該タイプが導出タイプである場合には、当該タイプを定義する`VTypeData`オブジェクトから名前が検索される。

`TypeHelp`

`VcrHelp *TypeHelp (VTypeCode)`

`type` それについての情報を入手しようとするタイプ。

`TypeHelp`は、指定されたタイプに関するヘルプ情報を検索する。当該タイプが導出タイプである場合には、当該タイプを定義する`VTypeData`オブジェクトからヘルプオブジェクトが検索される。`NULL`は、基本タイプに関して戻される。

### Mixins

クラスレジストリエレメントは、全て、`mix in`をサポートすることができる。これらの`mix in`は、ベースクラスレジストリにおいて定義されていない付加機能性を提供

することができる。例えば、`mix in`は、例えば整数タイプに関するカラーエディタのようなタイプに関する特殊化された編集ツールをサポートすることができる。ビジュアルエッジ (`Visual Edge`) は、`mix in`に関する標準スーツ (アップル・イベント・スーツに類似する) を定義するために、相互運用性を強化する目的で、そのパートナーと共同作業する意向である。`mix in`は、それぞれ、一意的な1つのIDを持つ。あらゆる販売者は、クラスレジストリエントリに関して各自の`mix in`を定義し、そして、定義およびIDを発表することが出来る。他の販売者は、それらのシステムにおいて機能性を具体化することが出来る。標準`mix in`スーツを定義しようとする販売者は、IDのブロックをビジュアルエッジに請求することができる。

### クラスVcrQueryMixin

`VMixin`の共用サブクラス

`VcrQueryMixin`は、クラスレジストリ構造自体の中のオブジェクトを含むクラスレジストリに「見える」エントリについてクラスレジストリに質

問するために用いられる抽象クラスである。この`mix in`の詳細については未だ決定されていない。問合せ質問`mix in`定義は、あらゆる特定の問合せ言語から独立した質問を実施する能力を提供すると同時に、OQL (ODMG'93 オブジェクト問合せ言語) と同様に複雑な質問に対してどの質問を選定するかを可能にするために十分な詳細を提供することを意図する。質問拡張の上記とは別の目標は、質問があらゆる特定の質問言語（例えば、SQL、OQL、Apple Event オブジェクト規則子レコード等々）に容易に翻訳されることを可能にすることである。この能力は、各オブジェクトシステムが、例えば質問を行うために各質問の固有のメカニズムを使用するような方法において、質問`mix in`を実行することを可能にするはずである。デフォルトとしての具体化は、収集においてアイテムを列挙し、そして、これらアイテムを質問にマッチさせる（低速であろうとも）ために存在する。

質問`mix in`は、`VClassData`、`VAdapterNameSpace`、`VViewNameSpace`、及び、`VClassRegistry` オブジェクトにおいて発見することができる。

クラスレジストリ質問に関しては、オブジェクトシステムアダプタの具体化に際して、当該オブジェクトシステムが、継承的に、複雑な質問をサポートしない場合には、デフォルトとしての質問`mix in`は、ビュー及びアダプタに対して作成することができる。このデフォルトとしての`mix in`は、ネームスペース又はオブジェクトシステムにおける全てのクラスを列挙する。従って、質問を個々のアイテムにマッチさせる。これは低速質問を生成するが、正しい結果を戻す。

或る種の形の質問をサポートするオブジェクトクラスインスタンス（クラスレジストリにおいて記述済み）に関しては、この`mix in`のバージョンを`VClassData`に

付加することが出来る。こうすれば、システムとは独立した方法において、コードがインスタンス階層に質問することが可能になる。この`mix in`のデフォルトバージョンは、インスタンスの収集特性の内容を列挙するためのメカニズムが

存在する限り使用可能であるような機能を提供することができる。

### クラスVcrExposureMixin

VMixinの共用サブクラス

VcrExposureMixinは、クラス定義、及び、他のオブジェクトシステムからの具体化情報をこのオブジェクトシステム（すなわち、そのためにmixinが設計されたオブジェクトシステム）に明白に露出するために使用するためのmixinであるような抽象クラスである。

露出(exposure)mixinは、VAdapterNameSpace及びVClassRegistryオブジェクトにおいて利用可能である。VClassRegistryインプレメンテーションは、各アダプタに関して同じオペレーションを実施する全てのアダプタを循環する。

更に、露出mixinは、外部オブジェクト（即ち、他のオブジェクトシステム内のオブジェクト）の周辺において固有プロキシオブジェクト（即ち、このオブジェクトシステム内のオブジェクト）を作成するためのサポートを提供する。これは、このオブジェクトシステム内のオブジェクトにとって、他のオブジェクトシステム内のオブジェクトがあたかもこのオブジェクトシステムに「固有」であるかのように見えることを可能にする。プロキシオブジェクトは、実オブジェクトの方法および特性をコールするためのVcrCall及びVcrCompleteCallにおけるクラスレジストリサポートを用いて、このオブジェクトシステムに対するコールを実オブジェクトに簡単にマップする。

露出mixinは、具体化情報をこのオブジェクトシステムに対して露出する方法を提供する。即ち、この機能性に含まれる特定のクラスのインスタンスを作成するためにどのプログラムを実行するべきか等々に関する情報は、「実行具体化（ランタイムインプレメンテーション）」リクエストを満足させるために作成された「もの」としてインスタンスを明白に識別する能力である。

オブジェクトシステムアダプタは、露出mixinに関して、幾つか又は全てをサポートできるか、或いは、一切サポートすることができない。オブジェクトシステムが任意のこの種機能性を実行する能力を提供しない場合には、アダプタにmixinを提供してはならない。

露出 `mix in` のサポートに加えて、オブジェクトシステムアダプタは、オブジェクトを透明に露出する能力を提供することも可能である。即ち、他のオブジェクトシステムからのオブジェクトがこのオブジェクトシステムの方法または特性にパスされる場合には、アダプタは、外部オブジェクトの周辺にプロキシオブジェクトを自動的に作成し、そして、

それを方法または特性にパスすることを選定することが可能である。クラスレジストリのユーザにとっては、透明に露出することによって、オブジェクトシステムが正当でなく統合されたように見えることを可能にする。透明な露出を使用すれば、クラスレジストリは、それらの間に一切の特殊変換を行うことなしに、インスタンスが1つのオブジェクトシステムからの他のオブジェクトシステムにパスすることを可能にする。

オーバーライド可能な共用メンバー

`ExposeDefinition` このオブジェクトシステムに対してクラスレジストリ定義を露出する。

`ExposeSubclass` あたかも、それがこのシステムのクラスの1つのサブクラスであるかのように、このオブジェクトシステムに対してクラスレジストリ定義を露出する。

`ExposeApplication` 特定のクラスのインスタンスを作成するためのソースとして、アプリケーションを露出する。

`ExposeFactory` このオブジェクトシステムからインスタンスを作成することができるようにランタイムにおいてクラスを露出する。

`ExposeInstance` ランタイムにおいてこのオブジェクトシステムからコール可能であるように、オブジェクトインスタンスを露出する。

`HideDefinition` 露出されたレジストリ定義をこのオブジェクトシステムから隠す。

`HideApplication` 特定のクラスのインスタンスを作成するた

(208)

めのソースとしてアプリケーションを隠す。

**HideFactory** インスタンスがこのオブジェクトシステムから作成されることがこの時点においては不可能であるように露出されたクラスファクトリを隠す。

**HideInstance** この時点においてはこのオブジェクトシステムからコールされることが不可能であるように、露出されたオブジェクトインスタンスを隠す。

**IsDefinitionExposed** 外部クラスがこのアダプタに既に露出済みであるかどうかを戻す。

**IsApplicationExposed** 特定のアプリケーションがこのアダプタに既に露出済みであるかどうかを戻す。

**AcquireProxy** 他のオブジェクトシステムから出たこのインスタンスに関して、このオブジェクトシステム内にプロキシ(proxy) オブジェクトを作成する。

**ReleaseProxy** 既存のプロキシオブジェクトを解放する。

**ConstructNameSpace** このオブジェクトシステム内にネームスペースを作成する。

**InstanceDeleted** オブジェクトインスタンスが削除済みであることをアダプタに通告する。

**UnmapForeign** 露出隠しに使用可能な外部アイテムに関する全てのリファレンスを除去する。

**UnmapAllForeign** 露出隠しに使用された外部アイテムに関する全てのリファレンスを除去する。

**ExposeDefinition**

```
status_t ExposeDefinition( VcrToplevel      *item,
                           VTToplevelRef    *exposed ) = 0
```

**item** このオブジェクトシステムに露出するためのアイテム。

**exposed** (戻し) このオブジェクトシステムにおいて発見される状態

(209)

におけるアイテム。

`ExposeDefinition`は、アイテムをこのオブジェクトシステムに露出する。アイテムが一旦露出されると、露出された戻しアーギュメントは、露出された定義に関するリファレンスによって、あたかもそれがこのオブジェクトシステムから出たリファレンスであるかのように、満たされなければならない。例えば、アイテムがOELクラス記述VOleClassDataであって、このオブジェクトシステムがDSOMである場合、`exposed`は、クラスをDSOMクラスとして表すVSOMClassDataでなくてはならない。アイテムが既に露出済みである場合、この方法は成功し、そして、露出されたアイテムを戻さねばならない。定義の露出がサポートされない場合には、この方法はNotSupportedStatusを戻さねばならない。露出が成功した場合には、OkStatusを戻さねばならない。

`ExposeSubclass`

```
status_t ExposeSubclass( VClassData    *cls,
                        VClassData    *super,
                        VClassRef      *exposed) =0
```

`cls` このオブジェクトシステムに露出するためのクラス。

`super` `cls`のサブクラスとして使用するためのスーパークラス。

`exposed` (戻し) このオブジェクトシステムにおいて発見された状態におけるクラス。

`ExposeSubclass`は、あたかも`cls`が`super`のサブクラスであるかのように`cls`をこのオブジェクトシステムに露出する。`super`は、このオブジェク

トシステムからのクラスでなければならない。`cls`は、それ自体のバージョンを用いて、`super`の全ての方法を具体化し直していなければならない。そうでない場合には、このコールは失敗する可能性がある。外部クラスを固有クラスのサブクラスとして露出すると、他のオブジェクトシステムからのクラスを、あたかもそれがこのオブジェクトシステムからのクラスの簡単なサブクラスである

(210)

かのように見せることが可能である。この機能性はアプリケーションにおいて使用可能であるか、或いは、`mix in`フレームワークは、抽象クラス及び`mix ins`を任意のオブジェクトシステムにおいて具現化することを可能にする。`cls`が既に露出済みである場合には、この方法は、成功し、そして、露出されたアイテムを戻さねばならない。外部クラスをサブクラスとして露出することがサポートされない場合には、この方法は`Not Supported Status`を戻さねばならない。露出が成功した場合には、`Ok Status`が戻されなければならない。

### `ExposeApplication`

```
status_t ExposeApplication( VClassData *exposed,
                             const char *appName
                             cost char *appFile,
                             cost char *hostName
                             bool t *multiple ) = 0
```

`exposed` このオブジェクトシステムに露出されるべきオブジェクトのクラス。

`appName` アプリケーションの名前。

`Filename` 実行可能なアプリケーションのファイルネーム。

`hostName` そこにおいてアプリケーションが実行されなければならないホスト。

`multiple` 同時に、特に命名された多重ホストにおいて実行するために、このアプリケーションが露出されることを可能にする。

`ExposeApplication`は、このオブジェクトシステムに対して、アプリケーションを露出する。このコールは、`exposed`のインスタンスを作成するために、このアプリケーションを実行することが可能であることを指定する。`exposed`は、`ExposeDefinition`又は`ExposeSubdass`によって戻された`VClassData`であってはならず、その代りに、オリジナルの`VClassData`でなくてはならない。`appName`は、このシステム登録ファシリティにおいて使用されるべきアプリケーションの読み取り可能な名前を指定する。`hostName`変数の異なる値は3つの

(211)

異なるビヘイビアに帰着可能である：

\* アプリケーションが特定のホストにおいてこのオブジェクトシステムによって実行されなければならない場合には、ホストネームは `hostName` において指定されなければ

ならない。

\* アプリケーションが、ホストが作成するあらゆる場所において局所的に実行されなければならない場合には、リクエスト `hostName` は `NUL` でなくてはならない。この場合、`multiple` は無視される。

\* アプリケーションが常に現行ホストにおいて実行されなければならない、そして、オブジェクトが他のホストから要請される場合には、`hostName` は、1つの単一期間（即ち、「.」）を含むストリングでなくてはならない。

アプリケーションが `multiple` により `TRUE` として登録された場合、`ExposeApplication` は、複数の異なるホストネームによってコールされることが可能である。これが許容される場合には、アプリケーションは、指定されたホストの任意の1つにおいて実行可能である。`multiple` が `FALSE` である場合には、異なるホストネームを用いたこの方法に対する各コールが最後の定義に置き換わるので、そこにおいてアプリケーションの実行が可能であるような特別に名指しされたホストが1つだけ存在する。

アプリケーションが既に露出済みである場合には、この方法は、正しいホスト及びファイルネームをセットするように定義を更新し、そして、`OkStatus` を戻さなくてはならない。この方法に対するコールによって行われたリクエストをアダプタがサポートしない場合には、`NotSupportedStatus` を戻さねばならない。露出が成功した場合には、`OkStatus` が戻されなければならない。

`ExposeFactory`

```
status_t ExposeFactory ( VClassData    *exposed,
                          void          *instance,
                          bool_t        mutiUse) = 0
```

(212)

`exposed` それに対してファクトリを露出するためのクラス。

`factory` (戻し) このオブジェクトシステムからのファクトリオブジェクト。

露出されたクラスのオブジェクトのソースであるように、`ExposeFactory`は現行プロセスを露出する。このファクトリは、プロセスの期間中のみ存在する。`exposed`は、`ExposeDefinition`又は`ExposeSubclass`によって戻された`VClassData`であってはならず、その代りに実クラスでなくてはならない。ファクトリが既に露出済みである場合において、この方法は、現行ファクトリを`factory`内に格納し、そして、`OkStatus`を戻さねばならない。この方法に対するコールによって行われるリクエストをアダプタがサポートしない場合には、`NotSupportedStatus`を戻さねばならない。露出が成功した場合には、`OkStatus`を戻さねばならない。

### `ExposeInstance`

```
status_t ExposeInstance ( VClassData      *exposed,
                          void             *instance
                          bool_t           *mutiUse ) = 0
```

`exposed` そのインスタンスを露出するためのクラス。

`instance` このオブジェクトシステムに露出するための`exposed`のインスタンス。

`mutiUse` このインスタンスが複数のリクエストによって使用可能であるかどうか。

`ExposeInstance`は、このオブジェクトシステムに対する`exposed`の特定のインスタンスを露出する。インスタンスは、当該アプリケーションに関するクラスの1つの「実行中インスタンス」として使用可能である。`exposed`は、`Exposeddefinition`又は`ExposeSubclass`によって戻された`VClassData`であってはならない。`exposed`及び`instance`は実オリジナル値でなくてはならない。この方法に

(213)

対するコールによって行われたリクエストをアダプタがサポートしない場合には、`NotSupportedStatus`を戻さねばならない。露出が成功した場合、或いは、インスタンスが既に露出済みである場合においては、`OkStatus`を戻さねばならない。この場合、`ExposeInstance`を用いて露出されたこのクラスの1つのインスタンスだけでなくてはならない。`multiUse`が`FALSE`である場合には、オブジェクトの使用が許容されるのはただ1人のクライアントに限られなければならない。`multiUse`が`TRUE`である場合には、複数のクライアントの全てが当該オブジェクトを使用することができる。

### HideDefinition

```
status_t HideDefinition (VcrToplevel *exposed) = 0
```

`exposed` このオブジェクトシステムに以前に露出されたアイテム。

`HideDefinition`は、このオブジェクトシステムに以前に露出されたアイテムを隠す。`HideDefinition`は、アイテムによって戻された`ExposeDefinition`及び`ExposeSubclass`に適用することができる。この方法に対するコールによって行われたリクエストを当該アダプタがサポートしない場合には、`NotSupportedStatus`を戻さねばならない。隠しが成功した場合には、`OkStatus`を戻さねばならない。

### HideApplication

```
status_t Hideapplication ( VClassData      *exposed,
                           const char      *appName,
                           const char      *appFile
                           const char      *hostName) = 0
```

`exposed` このオブジェクトシステムに既に露出されたオブジェクトのクラス。

`appName` アプリケーションの名前

`appFile` 実行可能なアプリケーションのファイルネーム。

`hostName` 実行するために指定されたアプリケーションのホスト。

(214)

`HideApplication`は、以前に露出されたアプリケーションを、このオブジェクトシステムから隠す。`appName`は、以前に`ExposeApplication`にパスされたとおりのアプリケーションの読み取り可能な名前を指定する。`hostName`変数の異なる値は、4種の異なるビヘイビアに帰着可能である：

- \* `hostName`が特定のホストネームである場合には、当該ホストにおいて実効するためのアプリケーション定義は除去されなければならない。
- \* `hostName`が`NULL`である場合には、当該アプリケーションの局所的に作動化不能な定義が隠される。
- \* `hostName`が「.」である場合には、現行ホストにおいて実効するためのアプリケーション定義は除去されなければならない。
- \* `hostName`が「\*」である場合には、全ての定義されたホストにおいて実行するためのアプリケーション定義は除去しなければならない。

この方法に対するコールによって行われるリクエストをサポートを当該アダプタがサポートしない場合には、`NotSupportedStatus`を戻さねばならない。隠しが成功した場合には、`OkStatus`を戻さなければならない。

### `HideFactory`

```
status_t HideFactory(VClassData *exposed) = 0
```

`exposed` 以前に露出されたファクトリのクラス。

`HideFactory`は、既に露出されたファクトリオブジェクトを隠す。`exposed`は、`ExposeFactory`に対するコールにおいて指定されていなければならない。この方法に対するコールによって行われたリクエストを当該アダプタがサポートしない場合には、`NotSupportedStatus`を戻さねばならない。隠しが成功した場合には、`OkStatus`を戻さねばならない。

### `HideInstance`

```
status_t HideInstance(void *instance) = 0
```

`instance` 固有オブジェクトシステムに以前に露出されたインスタン

(215)

ス。

`HideInstance`は、以前に露出されたインスタンスを固有オブジェクトシステムから隠す。`instance`は、`ExposeInstance`にパスされたアイテム

ムでなくてはならない。この方法に対するコールによって行われたリクエストを当該アダプタがサポートしない場合には、`NotSupportedStatus`を戻さねばならない。隠しが成功した場合には、`OkStatus`が戻されなければならない。

`IsDefinitionExposed`

```
bool_t IsDefinitionExposed (VcrToplevel *item) = 0
```

`item` このオブジェクトシステムに対する露出に関してチェックするためのアイテム。

`IsDefinitionExposed`は、このアイテムが以前にこのオブジェクトシステムに対して露出されたかどうかをチェックする。このコールは、コードがその露出が以前に行われたかどうかをチェックすることを可能にする。アイテムが以前に露出済みである場合には、露出を繰り返す必要はない。

`IsApplicationExposed`

```
bool_t IsApplicationExposed (      VClassData      *exposed,
                                  const char         *appName
                                  const char         appFile ) = 0
```

`exposed` チェックするためのオブジェクトのクラス。

`appName` アプリケーションの名前。

`appFile` 実行可能なアプリケーションのファイルネーム。

`IsApplicationExposed`は、このクラス及びアプリケーションが以前にこのオブジェクトシステムに露出されたかどうかをチェックする。このコールは、コードの露出が以前に行われたかどうかをコードがチェックすることを可能にする。アプリケーションが以前に露出済みである場合には、露出を繰り返す必要はない。

(216)

## AcquireProxy

```

status_t AcquireProxy( Vclassdata    *exposed,
                       void          *instance,
                       void          **proxy) = 0

```

`exposed` そのためにプロキシ (proxy:代理) を作成するためのクラス。

`instance` そのためにプロキシを作成するための `exposed` のインスタンス。

`proxy` (戻し) インスタンスを表す代理オブジェクト。

`AcquireProxy` は、このオブジェクトシステムのオブジェクトとして `proxy` を作成する。この `proxy` は、このオブジェクトシステムに対する `exposed` の特定のインスタンスを表す。この方法に対するコールによって行われたリクエストを当

該アダプタがサポートしない場合には、`NotSupportedStatus` を戻さねばならない。代理作成が成功するか、または、インスタンスが先在している代理を持つ場合には、`OkStatus` を戻さねばならない。代理は、`ReleaseProxy` によって処理されなければならない。

## ReleaseProxy

```

status_t ReleaseProxy (void *proxy) = 0

```

`proxy` 他のクラスのインスタンスを表すプロキシオブジェクト。

`ReleaseProxy` は、`AcquireProxy` によって以前に検索されたプロキシを解放する。この方法に対するコールによって行われたリクエストを当該アダプタがサポートしない場合には、`NotSupportedStatus` を戻さねばならない。プロキシの解放が成功した場合には、`OkStatus` を戻さなければならない。

## ConstructNameSpace

(217)

`status_t`

```
const char      *spaceName,
VTAdapterRef    *namespace)= 0
```

`spaceName` 新規なネームスペースの名前。

`namespace` (戻し) 新規に作成されたネームスペース。

`ConstructNamespace`は、新規なネームスペースによって作成された現行スペースを作成する。このネームスペースは、明白に露出されたクラスを格納するための場所として使用することが出来る。この方法に対するコールによって行われたリクエストを当該アダプタがサポートしない場合には、`NotSupportedStatus`を戻さねばならない。ネームスペースの作成が成功するか、或いは、ネームスペースが既に存在している場合には、`OkStatus`を戻さなければならない。

`InstanceDeleted`

```
void InstanceDeleted(void *instance)= 0
```

`instance` 削除されつつあるインスタンス。

`InstanceDeleted`は、インスタンスが削除されたことをアダプタに通告する。これは、透明な露出によって作成された全ての関連プロキシオブジェクトをアダプタが破壊することを可能にする。

`UnmapForeign`

```
void UnmapForeign(VcrToplevel *item)= 0
```

`item` このオブジェクトシステムに対して露出される可能性のあったアイテム。

`UnmapForeign`は、当該アダプタがアイテムに対して使用可能な全てのリファレンスを除去する。この方法は、定期的なアダプタキャッシュページが、長い実行サーバプロセスにおけるメモリ消費を制限することを可能にする。

`UnmapAllForeign`

```
void UnmapAllForeign(
    VTObjectSystem system=KVAnyobjectSystem)= 0
```

(218)

system これに関するリファレンスを除去するためのオブジェクトシステム。

UnmapAllForeignは、指定されたオブジェクトシステムからのアイテムに対してアダプタにより使用可能な全てのリファレンスを除去する。この方法は、定期的なアダプタキャッシュパージが、長い実行サーバプロセスにおけるメモリ消費を制限することを可能にする。

【図1】

「ネイティブ」オブジェクトシステムを使用してインプリメントされるプロセス1

「外部」オブジェクトシステム

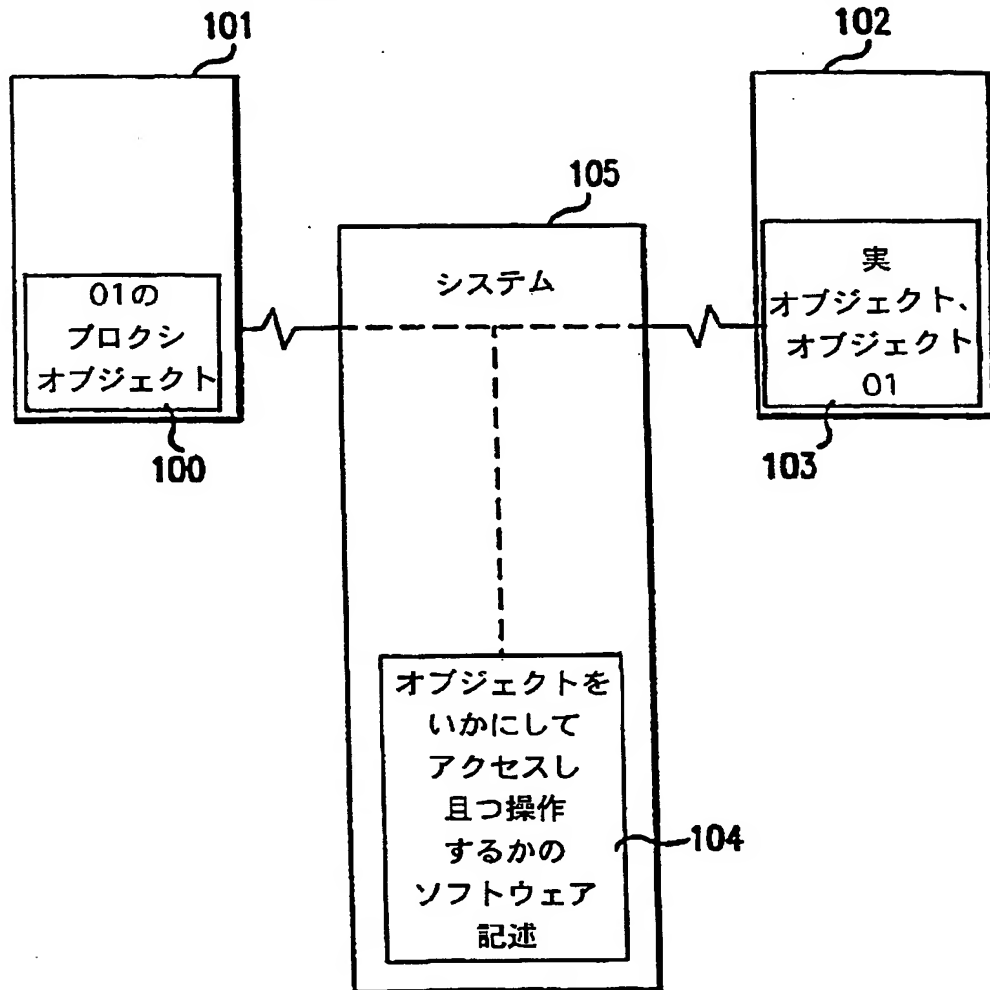


FIG 1

(219)

【図2】

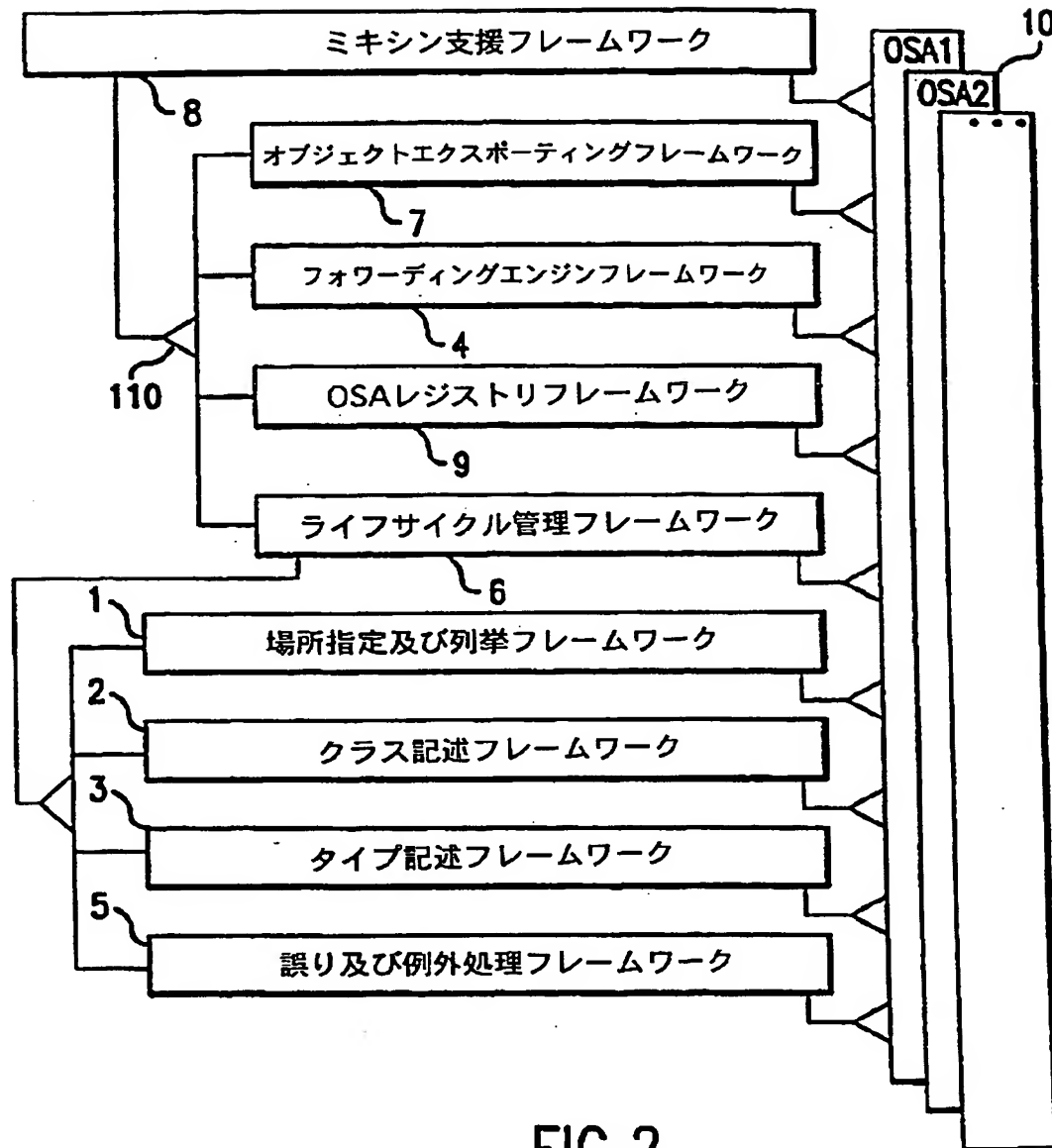


FIG.2

(220)

【図2】

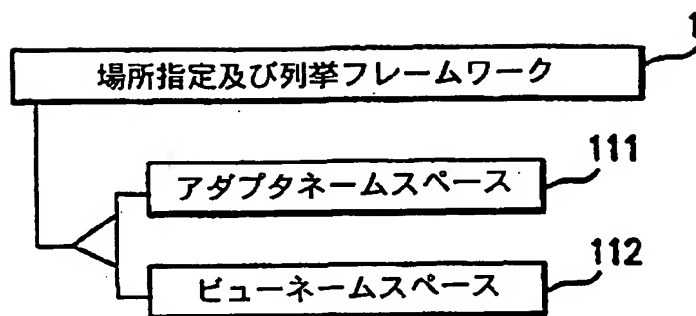


FIG.2b

【図3】

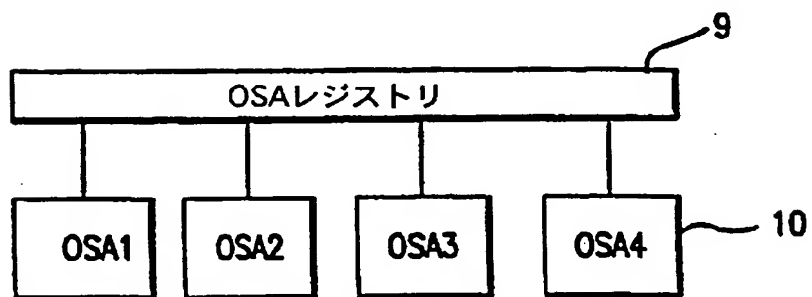
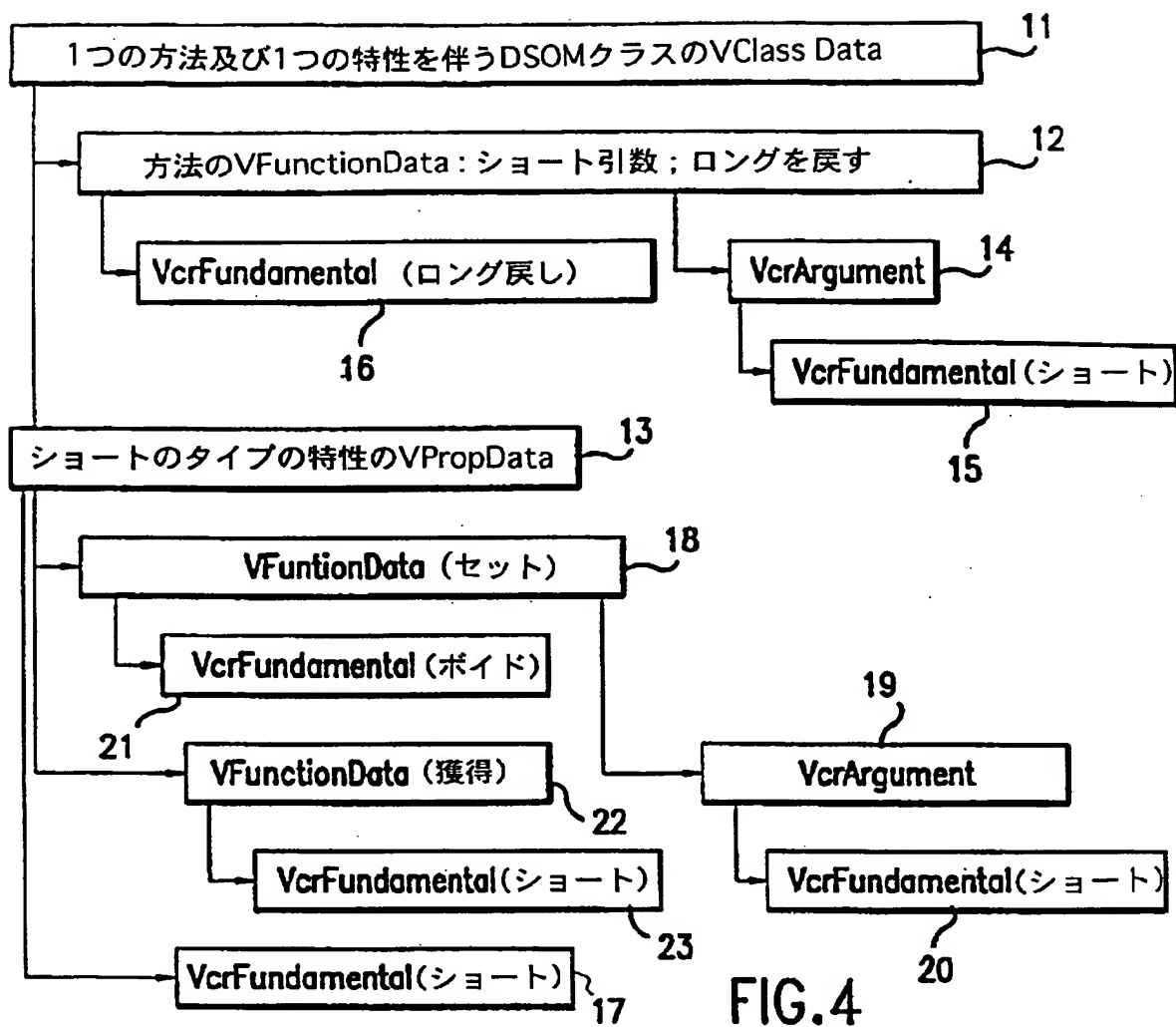


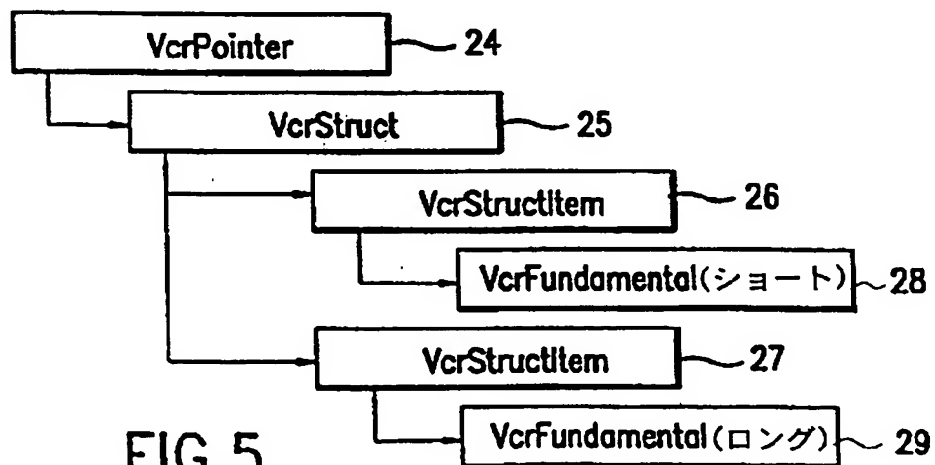
FIG.3

(221)

【図4】



【図5】



(222)

【図6】

34      35                                  30                  31      32      33  
 ↙      ↙                                  ↙                  ↙      ↙      ↙  
 void Classname\_Methodname (object,      pEnv,      X,      Y)

FIG.6a

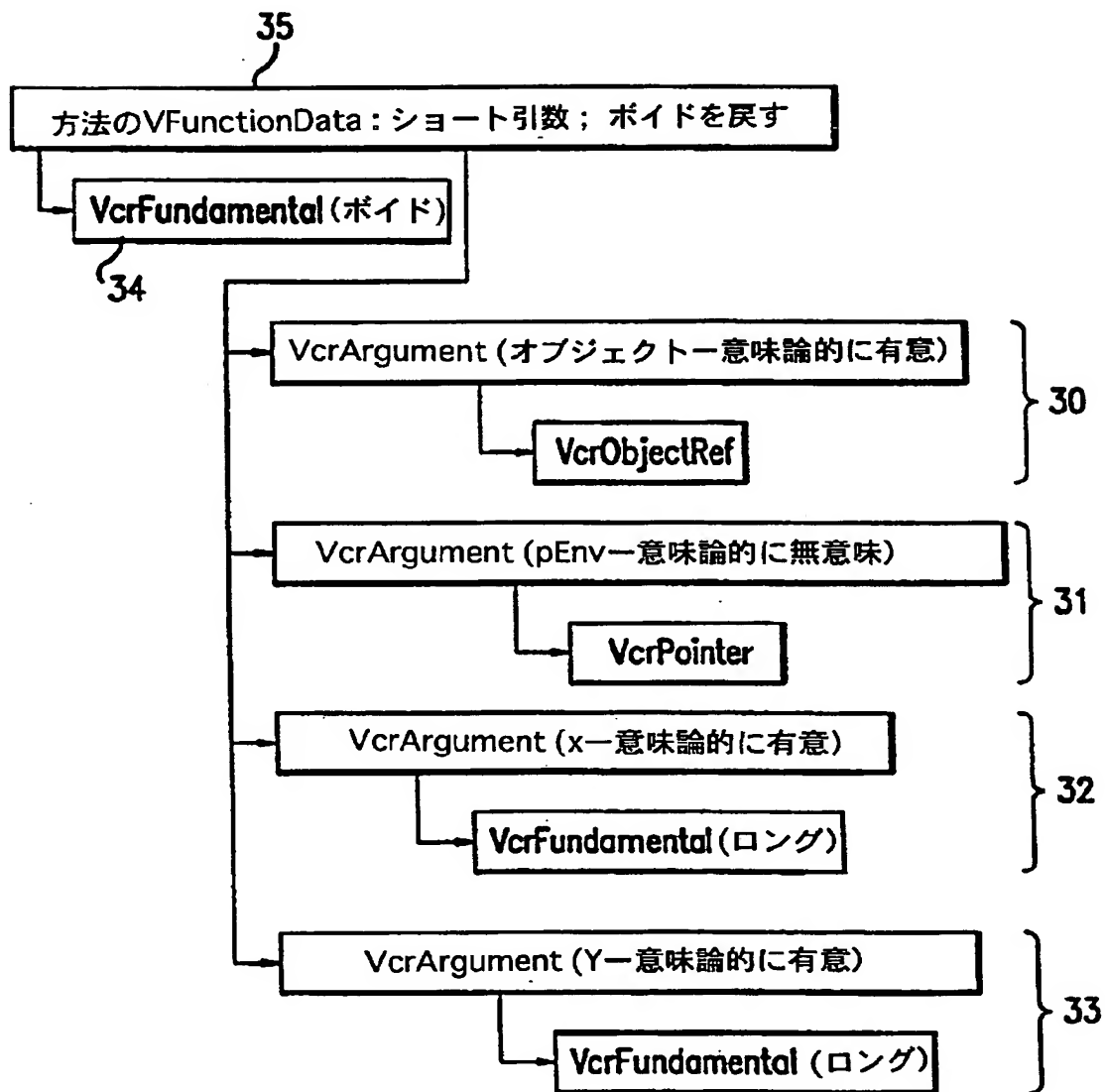
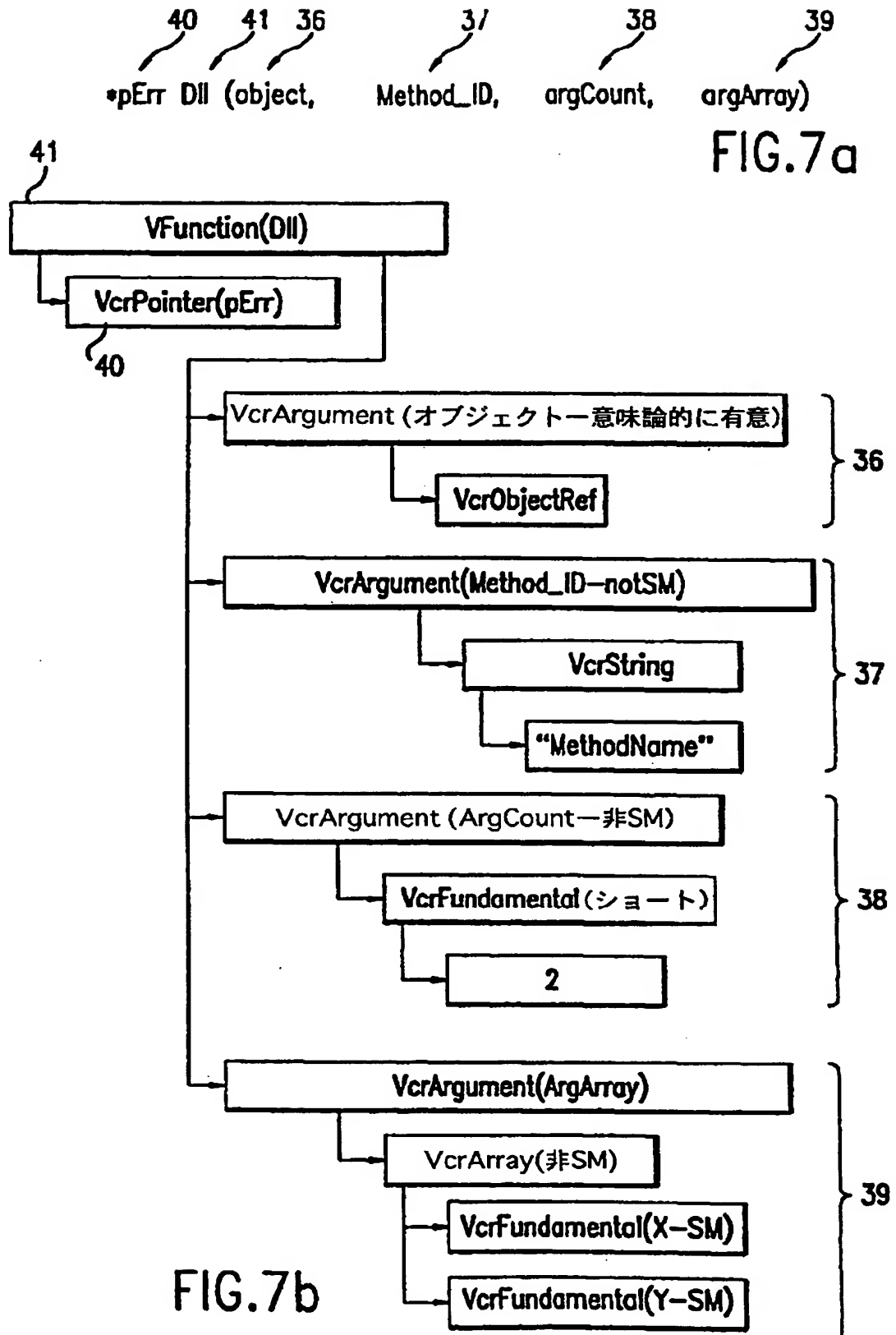


FIG.6b

(223)

【図7】



(224)

【図7】

呼出しソフトウェアから制御の流れを受取る。

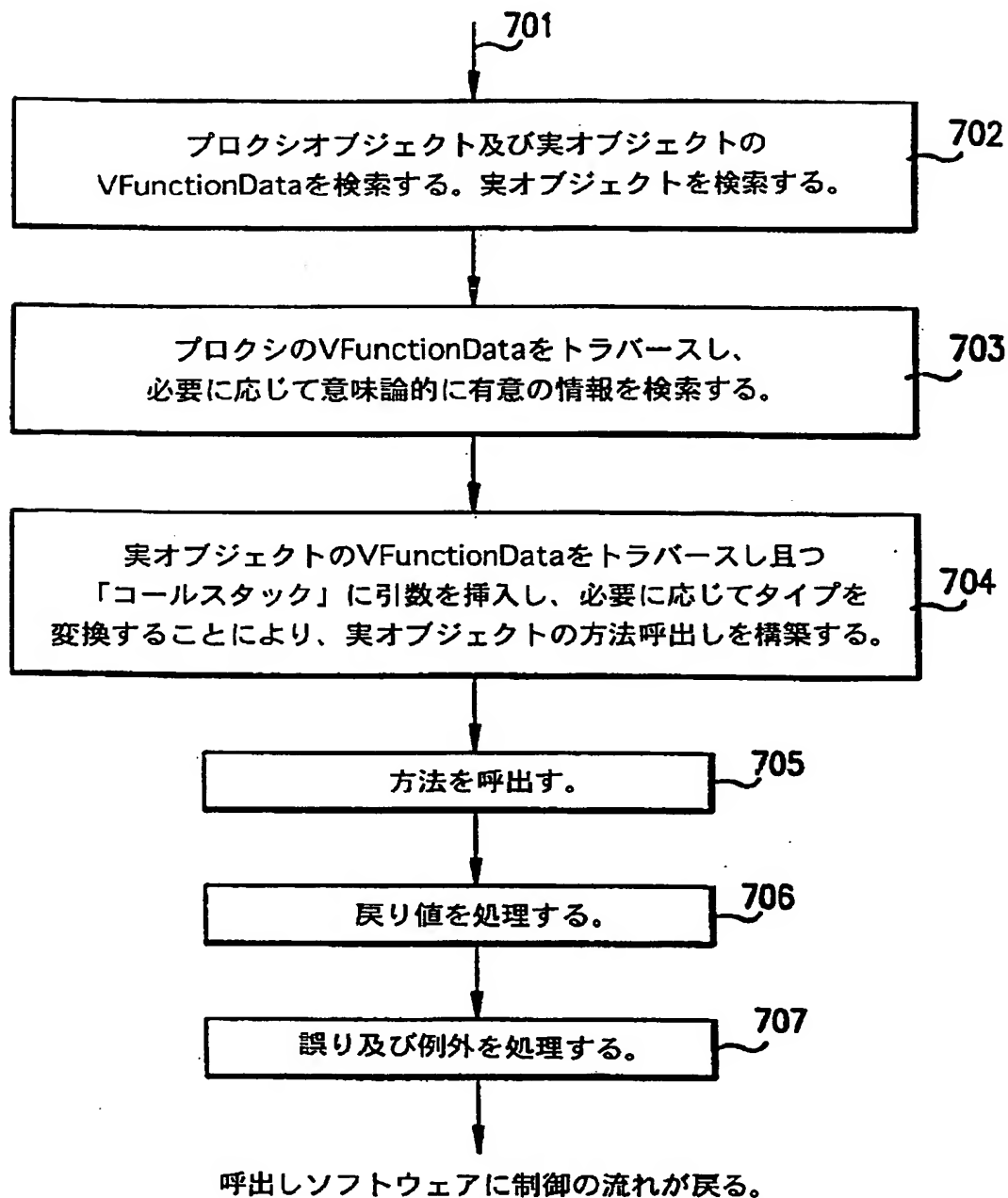


FIG.7c

(225)

【図8】

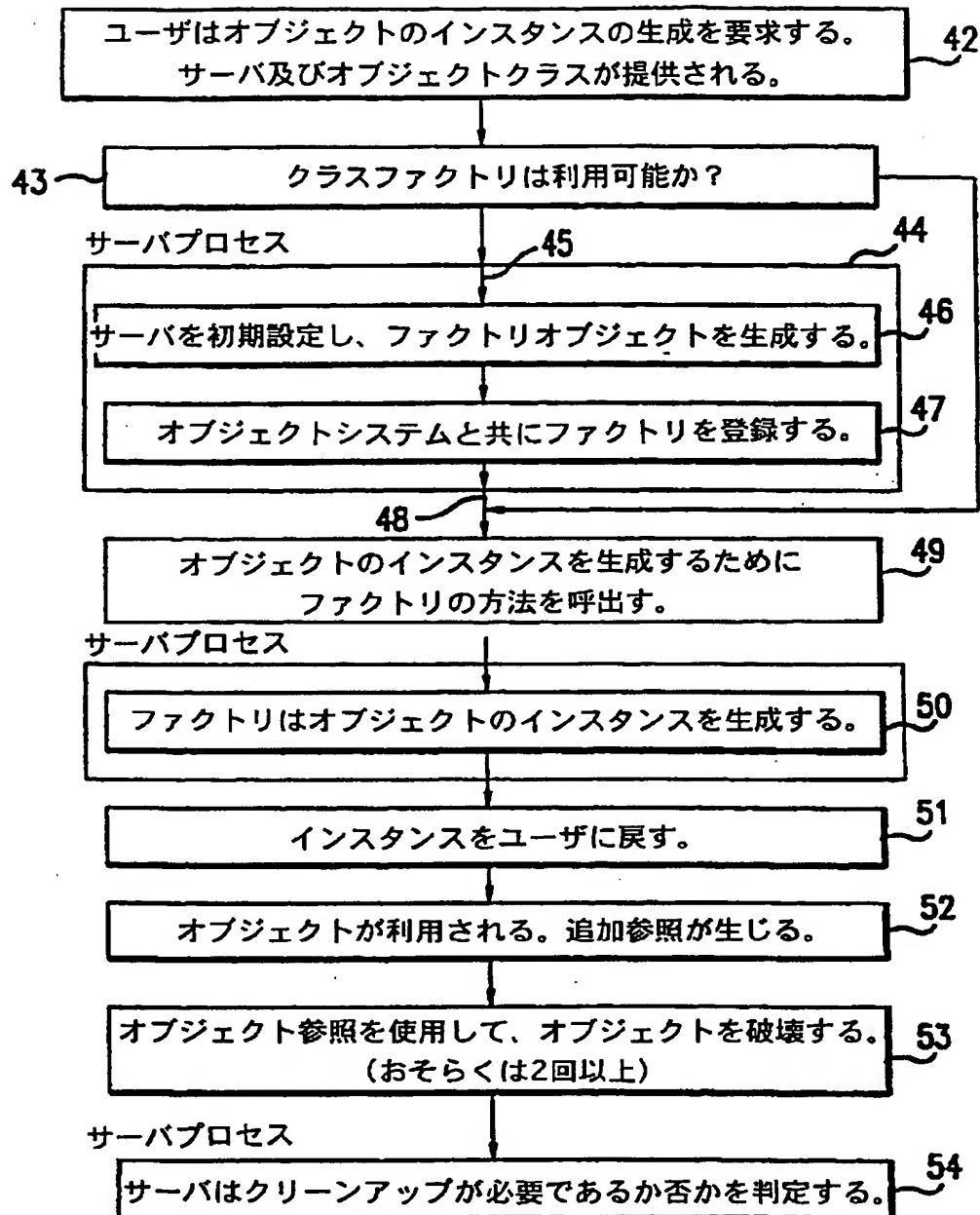


FIG. 8

(226)

【図9】

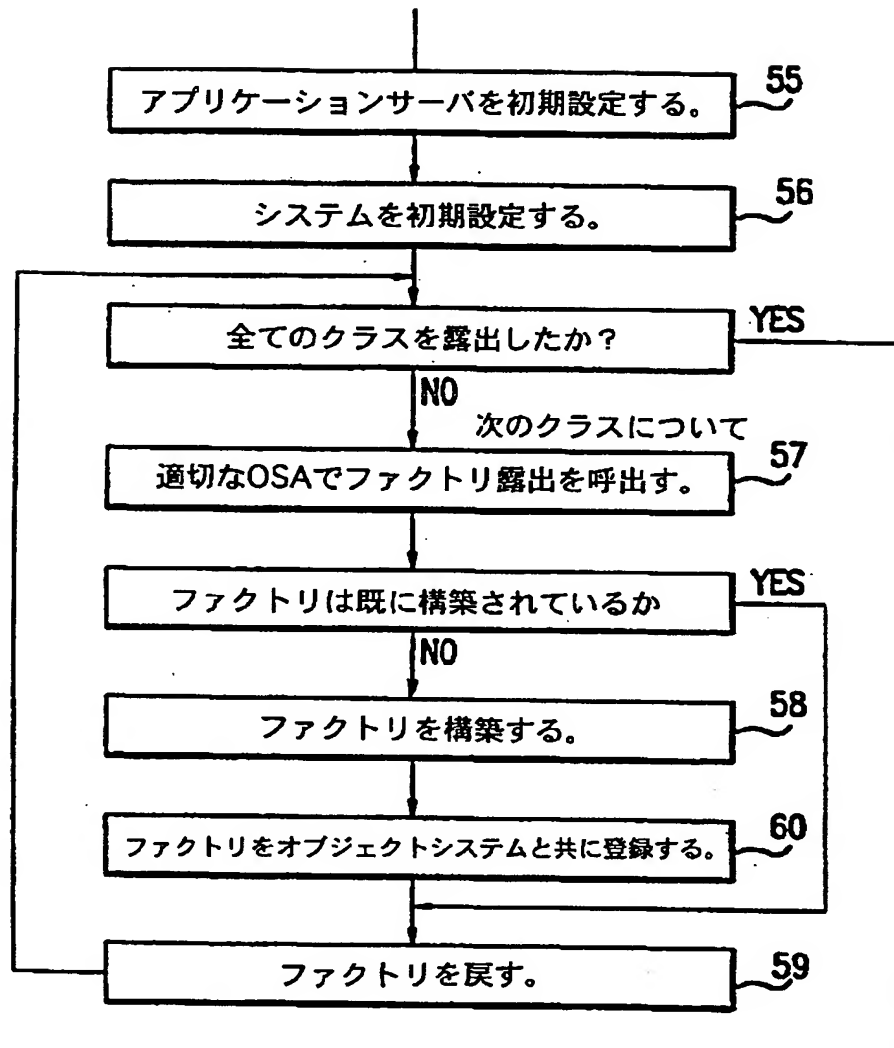


FIG.9

(227)

【図10】

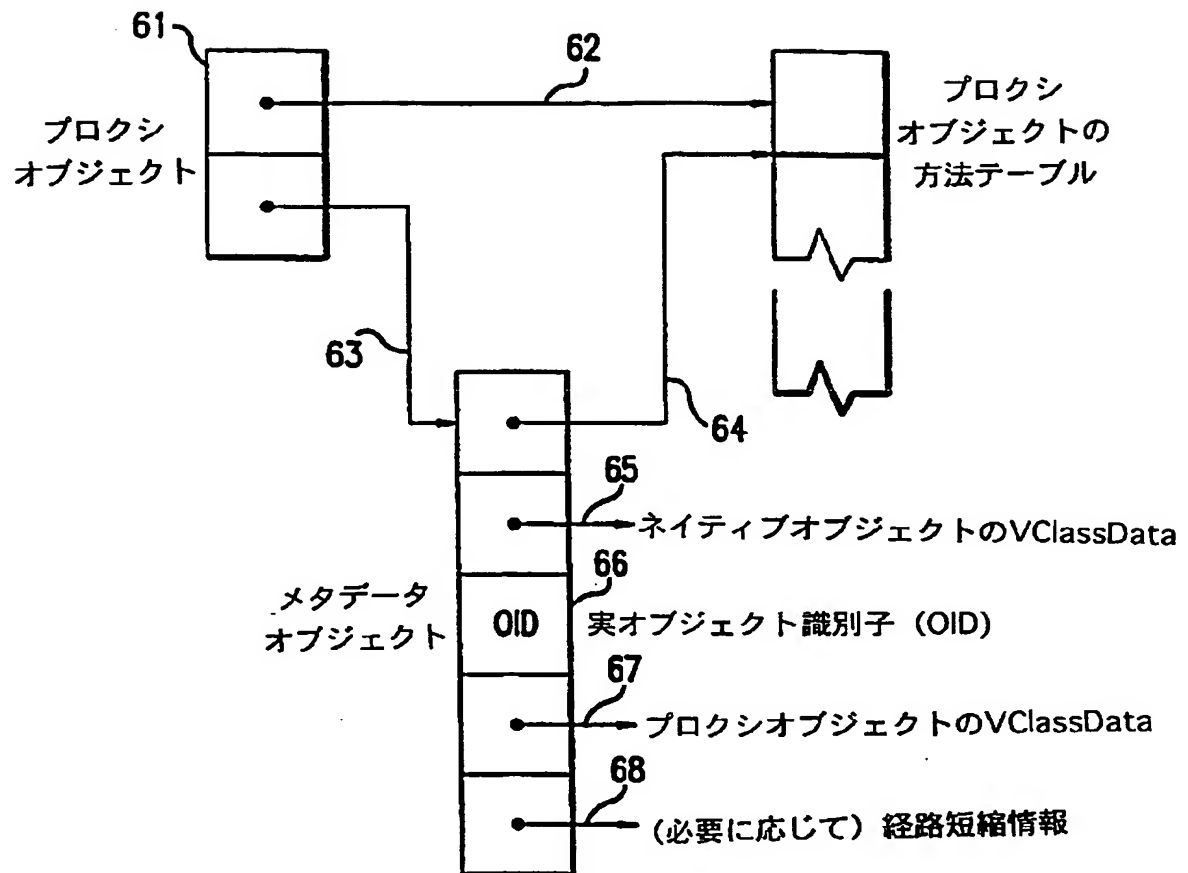


FIG.10

(228)

【図11】

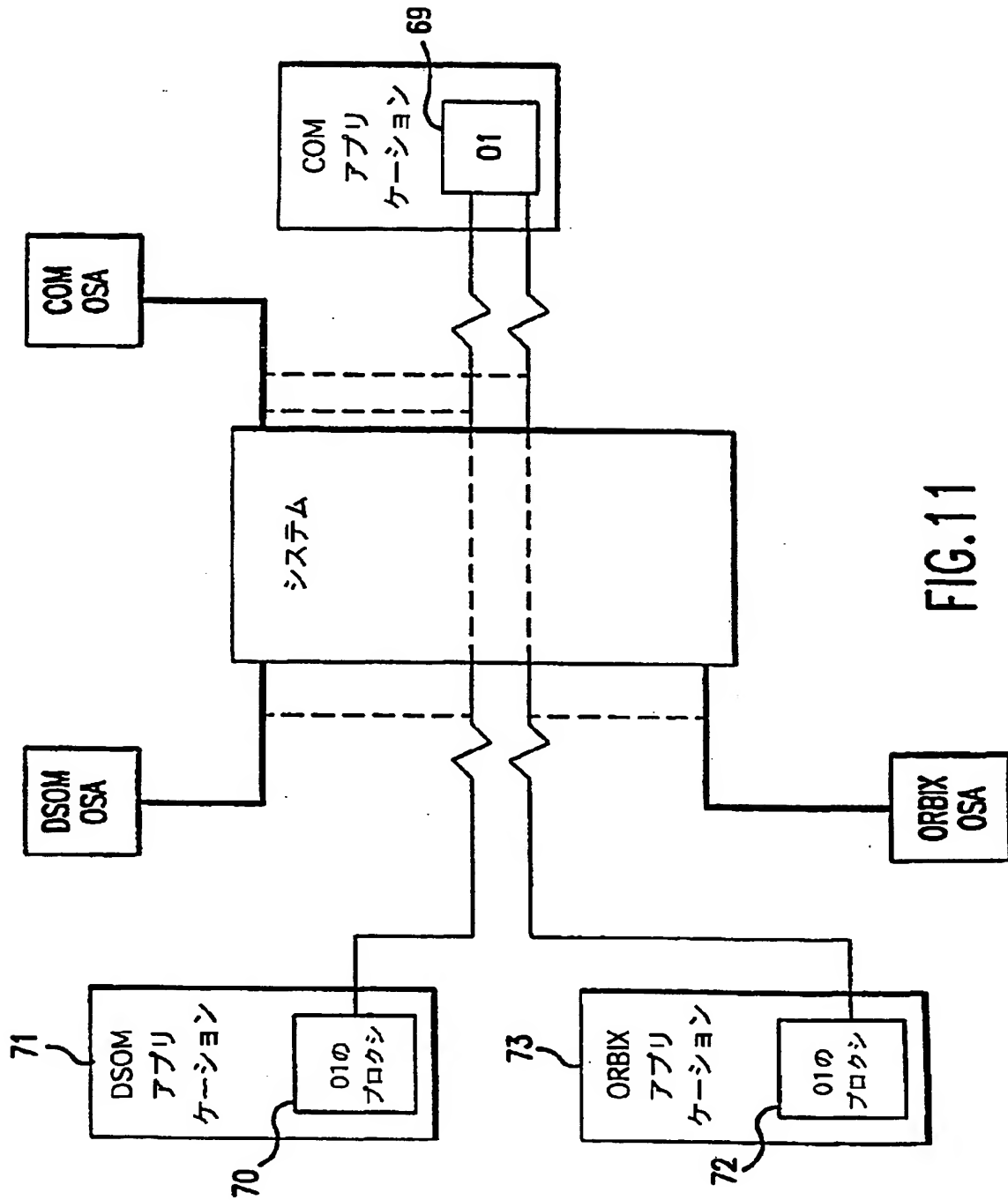
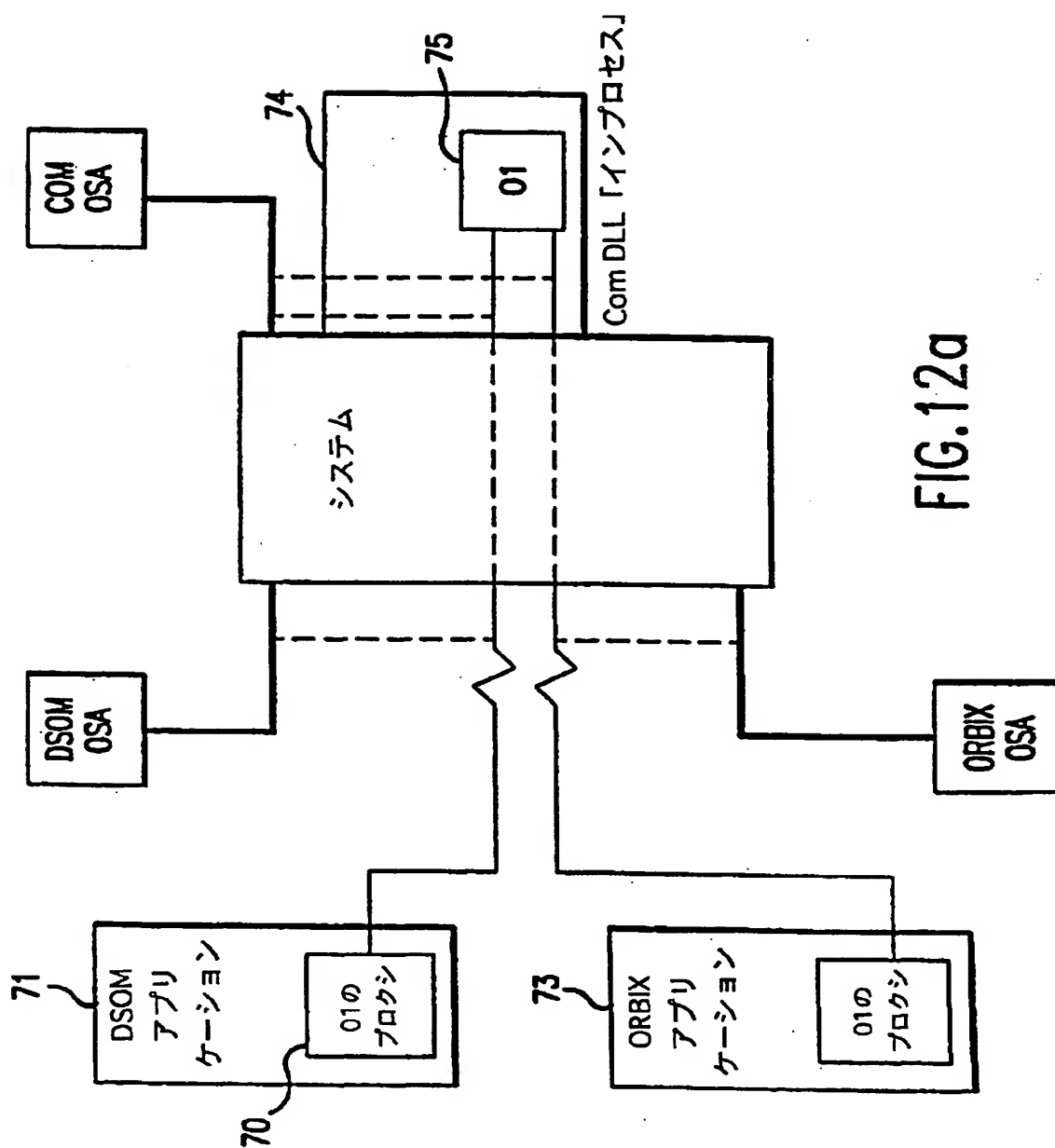


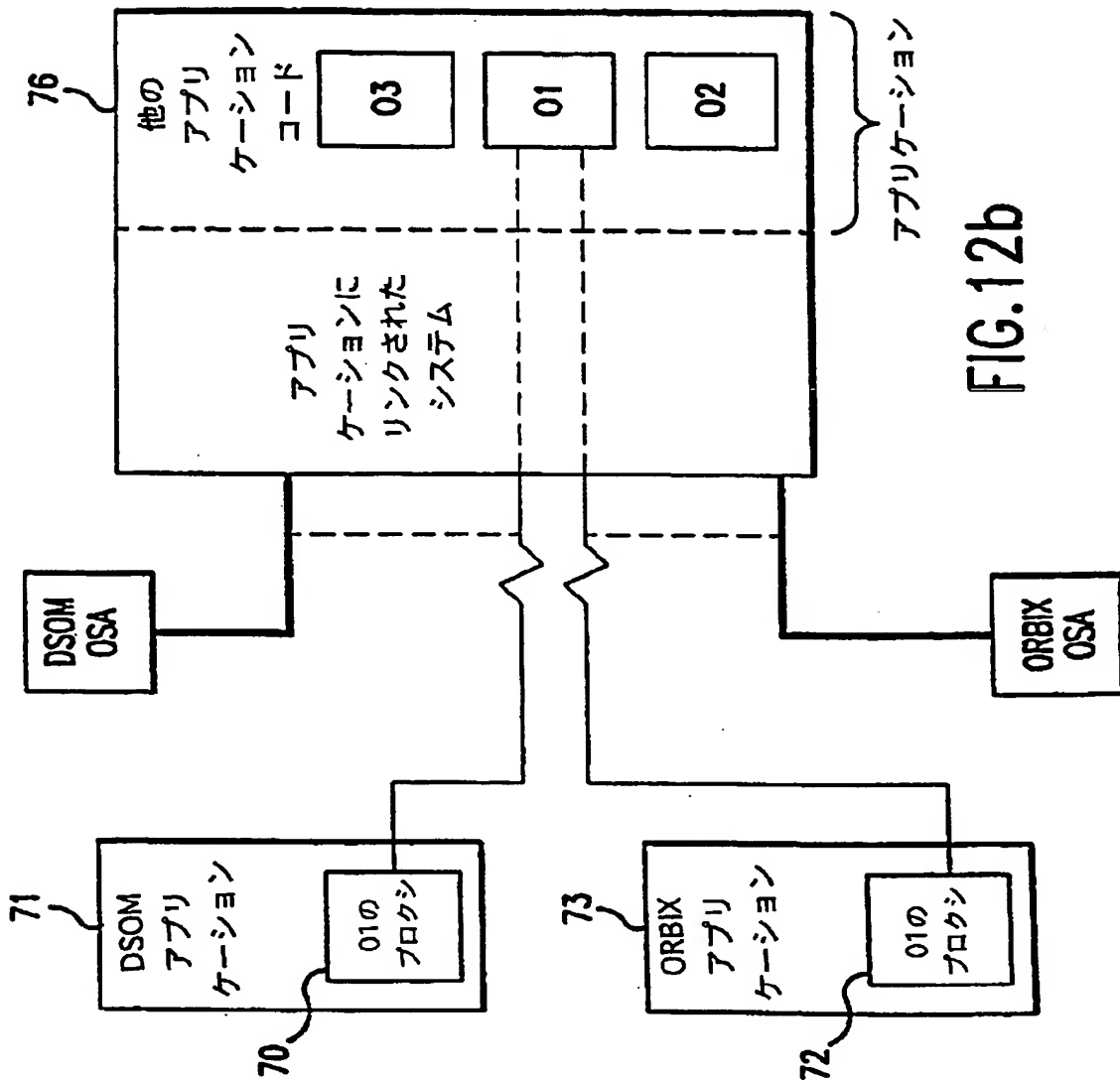
FIG.11

【図 12】



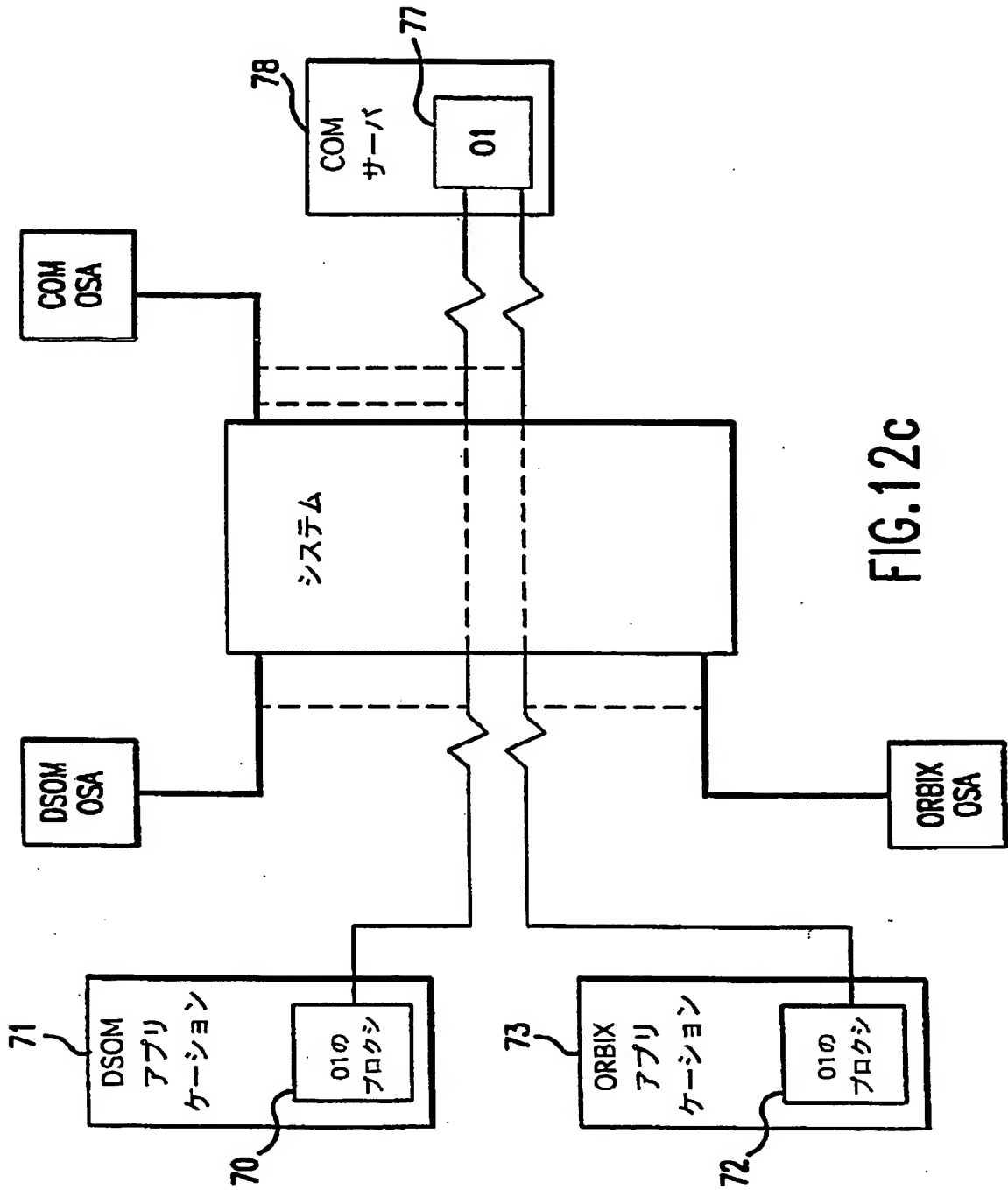
(230)

【図12】



(231)

【図12】



(232)

【国際調査報告】

<b>INTERNATIONAL SEARCH REPORT</b>		Inter. Appl. No. <b>PCT/CA 95/00513</b>
<b>A. CLASSIFICATION OF SUBJECT MATTER</b> <b>IPC 6 G06F9/44</b>		
According to International Patent Classification (IPC) or to both national classification and IPC		
<b>B. FIELDS SEARCHED</b> Minimum documentation searched (classification system followed by classification symbols) <b>IPC 6 G06F</b>		
Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched		
Electronic data base consulted during the international search (name of data base and, where practical, search terms used)		
<b>C. DOCUMENTS CONSIDERED TO BE RELEVANT</b>		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP,A,O 495 279 (IBM) 22 July 1992 see the whole document ---	1-23
A	PROCEEDINGS OF THE SECOND INTERNATIONAL WORKSHOP ON OBJECT ORIENTATION IN OPERATING SYSTEMS (CAT. NO.92TH0477-0), DOURDAN, FRANCE, 24-25 SEPT. 1992, ISBN 0-8186-3015-9, 1992, LOS ALAMITOS, CA, USA, IEEE COMPUT. SOC. PRESS, USA, pages 212-220, DAVE A ET AL 'Proxies, application interfaces, and distributed systems' see the whole document --- -/-	1-23
<input checked="" type="checkbox"/> Further documents are listed in the continuation of box C. <input checked="" type="checkbox"/> Patent family members are listed in annex.		
* Special categories of cited documents : "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier document but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "d" document member of the same patent family		
Date of the actual completion of the international search <b>19 December 1995</b>		Date of mailing of the international search report <b>03.01.96</b>
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+ 31-70) 340-2040, Telex 31 651 epo nl, Fax (+ 31-70) 340-3016		Authorized officer <b>Brandt, J</b>

(233)

## INTERNATIONAL SEARCH REPORT

Inter-  
national Application No  
PCT/CA 95/00513

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT		
Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 36, no. 8, August 1993 NEW YORK, US, pages 457-458, XP 000390292 ANONYMOUS 'Use of System Object-Model Objects from Dynamic Languages' see the whole document -----	1-23

(234)

## INTERNATIONAL SEARCH REPORT

**Information on patient family members**

Inter val Application No  
PCT/CA 95/00513

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP-A-0495279	22-07-92	JP-A- 4277838 JP-B- 6093227	02-10-92 16-11-94
-----			

(235)

---

フロントページの続き

(81) 指定国 EP(AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OA(BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), AP(KE, MW, SD, SZ, UG), AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, US, UZ, VN

(72) 発明者 フーディ, マイケル, エイ.

カナダ国 ケベック州 エイチ3イー 1  
エム3 ナンズ アイランド ガブリエル  
ロイ 21